UNIVERSITÉ DE MONTRÉAL

COMPARAISON DE MÉTHODES HEURISTIQUES POUR LE PROBLÈME D'HORAIRES DE VÉHICULES AVEC DÉPÔTS MULTIPLES

ANN-SOPHIE PEPIN
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES APPLIQUÉES)
DÉCEMBRE 2006

Your file Votre référence
ISBN: 978-0-494-25565-0
Our file Notre référence
ISBN: 978-0-494-25565-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

COMPARAISON DE MÉTHODES HEURISTIQUES POUR LE PROBLÈME D'HORAIRES DE VÉHICULES AVEC DÉPÔTS MULTIPLES

présenté par : PEPIN Ann-Sophie
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. LANGEVIN André, Ph.D., président
M. DESAULNIERS Guy, Ph.D., membre et directeur de recherche
M. HERTZ Alain, Doct. ès Sc., membre

Remerciements

Je remercie sincèrement M. Guy Desaulniers, mon directeur de recherche. Il a été à la fois présent et patient pour répondre à mes nombreuses questions. Je le remercie également pour l'aide financière qu'il m'a fournie pour la durée de mes études.

Merci également à M. Alain Hertz et M. Ahmed Hadjar pour leur aide sur la recherche taboue et la génération de colonnes respectivement, de même qu'à tous ceux qui m'ont conseillée et soutenue.

Je ne me serais pas rendue jusqu'ici sans Loïc qui m'a ouvert la voie et qui m'a aidé à plusieurs niveaux. Merci à mes parents d'avoir toujours su que j'étais capable.

Résumé

Ce mémoire porte sur le problème d'horaires de véhicules avec dépôts multiples (MDVSP). Ce problème consiste à effectuer, à moindre coût, un ensemble de tâches ayant une durée et une heure de départ fixées par une flotte de véhicules provenant de différents dépôts. Le but de ce mémoire est d'effectuer, pour ce problème, une comparaison de différentes approches heuristiques dont certaines sont basées sur des techniques de programmation mathématique et d'autres sont des métaheuristiques. Nous présentons également deux améliorations, l'une pour diminuer les temps de calcul de deux de ces méthodes et l'autre pour obtenir des solutions entières plus rapidement pour l'une de celles-ci.

Le MDVSP peut se formuler comme un problème de partitionnement d'ensemble avec contraintes supplémentaires pouvant être résolu par une approche heuristique de génération de colonnes. Il peut également être formulé comme un programme linéaire en nombres entiers. Cette dernière formulation a été utilisée pour résoudre le MDVSP avec une méthode basée sur la relaxation lagrangienne et un algorithme de séparation et évaluation progressive, tous deux heuristiques. Nous avons également implémenté et testé des méthodes de recherche taboue et de recherche à voisinage large.

Nos algorithmes ont été testés sur des données générées aléatoirement. Les résultats obtenus montrent que la méthode de génération de colonnes heuristique fournit des résultats de qualité supérieure aux autres méthodes. Par contre, les temps de calcul sont relativement longs. La recherche à voisinage large offre, quant à elle, de bonnes solutions rapidement, sans toutefois atteindre la qualité obtenue par génération de colonnes.

Le dernier chapitre est consacré aux différentes améliorations implémentées. La méthode de génération de colonnes ne fournit une solution entière qu'à la toute fin du processus. Nous avons cherché à obtenir, à chacun des noeuds de l'arbre de branchement, une solution entière. Cependant, la qualité des solutions obtenues pour les premiers noeuds de l'arbre de branchement n'est pas très bonne et cette technique augmente le temps de calcul total.

Une autre amélioration vise à accélérer les temps de calcul pour les méthodes de génération de colonnes et séparation et évaluation progressive. En éliminant les arcs de coût élevé dès le début, le problème devient plus petit et donc plus rapide à résoudre. En conservant pour chaque noeud un maximum de $n/3$ (resp. $n/5$ et $n/6$) arcs inter-tâches pour les instances de $n = 500$ (resp. 1000 et 1500) tâches, nous pouvons obtenir des solutions dont la qualité ne s'est pas détériorée de plus de 0,05% de sa valeur. Le gain de temps varie de 24,1% à 34,8% pour la résolution par génération de colonnes et de 31,2% à 59,6% pour l'algorithme de séparation et évaluation progressive.

Abstract

This master thesis considers the multiple depot vehicle scheduling problem (MDVSP). This problem consists in covering, at the lowest cost, a set of tasks with fixed duration and departure time by a fleet of vehicles assigned to multiple depots. The purpose of this thesis is to make a comparison of various heuristic approaches for this problem. Some of them are based on mathematical programming decomposition techniques, while others are metaheuristics. We also present two improvements, one to decrease computational time for two of the methods and another to obtain integer solutions more quickly for one method.

The MDVSP can be formulated as a set partitioning problem with side constraints which can be solved by a heuristic column generation approach. It can also be formulated as an integer linear program. This last formulation was used to solve the MDVSP with a Lagrangian heuristic and a heuristic branch-and-bound algorithm. We also implemented and tested a tabu search heuristic and a large neighborhood search heuristic.

Our algorithms were tested on randomly generated data. The results obtained show that the heuristic column generation method provides better results than the other methods. Computational times are however relatively long. On the other hand, the large neighborhood search method generates good solutions quickly, without reaching the solutions quality obtained by column generation.

The last chapter is dedicated to the implemented improvements. The column generation heuristic finds an integer solution only at the end of the solution process. We tried to obtain, at each node of the branching tree, an integer solution. However,

solutions quality for the first nodes of the branching tree is poor and this technique inceases total computational time.

Another improvement aims at accelerating computational times for the column generation and branch-and-bound methods. By eliminating high cost arcs a priori, the problem becomes smaller and thus faster to solve. If we keep, for every node, at most $n/3$ (resp. $n/5$ and $n/6$) inter-task arcs for the instances involving $n = 500$ (resp. 1000 and 1500) tasks, we can obtain solutions that are not worse than 0,05% from the original value. Gains in solution time vary from 24,1% to 34,8% for the column generation heuristic and from 31,2% to 59,6% for the branch-and-bound algorithm.

Table des matières

Liste des tableaux

Liste des figures

Introduction

Le problème d'horaires de véhicules avec dépôts multiples (MDVSP pour multiple depot vehicle scheduling problem) consiste à effectuer, à coût minimum, un ensemble de tâches par un ensemble de véhicules répartis en plusieurs dépôts. Ce problème peut modéliser la planification des horaires d'autobus d'une ville, où les tâches représentent des trajets ou encore la planification des horaires de véhicules d'une entreprise spécialisée dans le transport de cargaisons monopolisant tout un camion où les tâches représentent le déplacement d'une telle cargaison. Ce problème est encore étudié aujourd'hui car les instances sont de plus en plus grandes à résoudre et nous amènent à développer de nouvelles méthodes de résolution de plus en plus rapides et efficaces.

Le but premier de ce mémoire est d'effectuer, pour le MDVSP, une comparaison de différentes approches heuristiques dont certaines sont basées sur des techniques de programmation mathématique et d'autres sont des métaheuristiques. Il s'agit d'une première dans le sens où, à notre connaissance, ces deux classes d'heuristiques n'ont jamais été comparées sur le même problème, avec des instances identiques et dans un même environnement de tests.

La structure de ce mémoire est la suivante. Nous définissons tout d'abord le problème dans le premier chapitre. Nous y présentons également les formulations que nous utilisons. Le générateur d'instances est décrit suivi d'une revue de littérature et des contributions de ce mémoire.

Dans le deuxième chapitre, nous introduisons les cinq méthodes testées : génération de colonnes heuristique, algorithme de séparation et évaluation progressive heuristique, heuristique lagrangienne, recherche taboue et recherche à voisinage large. Nous

fournissons également les résultats que nous avons obtenus sur des instances de 4 et 8 dépôts avec 500, 1000 et 1500 tâches, de même que les conclusions que nous avons pu établir.

Dans le troisième chapitre, nous exposons deux améliorations que l'on a développées, l'une pour diminuer les temps de calcul (pour les méthodes de génération de colonnes et de séparation et évaluation progressive) et l'autre pour obtenir des solutions entières plus rapidement (pour la méthode de génération de colonnes). Nous fournissons également des résultats sur ces améliorations.

Chapitre 1 : Présentation du problème

Nous commencons ce chapitre en définissant le problème d'horaires de véhicules avec plusieurs dépôts (MDVSP). Nous présentons ensuite les deux formulations mathématiques de ce problème qui seront utilisées, suivies de la description des instances. Nous terminons par une revue de littérature et les contributions de ce mémoire.

1.1 Description du MDVSP

Le MDVSP consiste à effectuer, à coût minimum, un ensemble de n tâches $N = \{1, 2, ..., n\}$ par une flotte de véhicules répartis dans un ensemble de m dépôts $K = \{1, 2, ..., m\}$. La tâche i doit commencer au temps a_i et sa durée est de d_i. Chaque tâche possède également un lieu de départ et un lieu de fin. Finalement, nous ajoutons une contrainte de capacité sur les dépôts, c'est-à-dire que chaque dépôt k possède au maximum v_k véhicules. Une tournée est une suite de tâches effectuées par un même véhicule et qui possède les caractéristiques suivantes :

- une tournée doit commencer et se terminer au même dépôt ;
- deux tâches i et j peuvent se succéder si $a_i + d_i + t_{ij} \leq a_j$ où t_{ij} est le temps nécessaire pour se rendre de la fin de la tâche i au début de la tâche j.

Si la relation précédente donne une inégalité stricte, le véhicule est considéré en avance et peut attendre jusqu'au temps a_j pour commencer la tâche j. Dans ce type de problème, une affectation valide des tâches implique que chaque tâche soit couverte

par exactement une tournée (un véhicule). On cherche donc à minimiser le coût total, composé des coûts associés aux tournées. Le coût d'une tournée comprend un coût fixe c_{fixe} pour le véhicule et un coût de déplacement.

1.2 Formulations mathématiques

Deux types de formulation du MDVSP sont présentées dans cette section. La première, une formulation basée sur les chemins et de type partitionnement d'ensemble avec contraintes supplémentaires, sera utilisée par l'approche de génération de colonnes. La deuxième, une formulation basée sur les arcs ou formulation multi-commodités en nombres entiers, servira à définir le réseau pour l'heuristique lagrangienne et pour l'algorithme de séparation et évaluation progressive heuristique.

1.2.1 Formulation basée sur les chemins

Cette formulation, présentée par Ribeiro et Soumis (1994), définit une variable pour chaque chemin possible. Cette formulation fait appel à la notation suivante :

K : l'ensemble des dépôts ;

Ω^k : l'ensemble des tournées possibles pour le dépôt $k \in K$ ($p \in \Omega^k$ est une tournée valide) ;

c_p^k : le coût de la tournée p associée au dépôt k ;

θ_p^k : une variable binaire qui vaut 1 si la tournée p associée au dépôt k a été choisie, 0 sinon ;

a_{ip}^k : un paramètre binaire qui vaut 1 si la tournée p associée au dépôt k contient la tâche i, 0 sinon.

En utilisant cette notation, le MDVSP peut se formuler comme suit :

$$Min \sum_{k \in K} \sum_{p \in \Omega^k} c_p^k \theta_p^k \tag{1.1}$$

sujet à :

$$\sum_{k \in K} \sum_{p \in \Omega^k} a_{ip}^k \theta_p^k = 1, \qquad \forall i \in N \tag{1.2}$$

$$\sum_{p \in \Omega^k} \theta_p^k \leq v^k, \qquad \forall k \in K \tag{1.3}$$

$$\theta_p^k \in \{0, 1\}, \qquad \forall p \in \Omega^k, \forall k \in K. \tag{1.4}$$

La fonction objectif (1.1) cherche à minimiser la somme des coûts de toutes les tournées utilisées. Les contraintes (1.2) imposent que chaque tâche soit présente dans une et une seule tournée et les inégalités (1.3) limitent le nombre de tournées par dépôt.

Ce modèle est un programme linéaire en nombres entiers qui comporte un très grand nombre de variables (une par tournée réalisable). Pour des instances de grande taille, il peut être résolu par une méthode de génération de colonnes imbriquée dans une procédure heuristique de séparation et évaluation progressive, semblable à celle de Desaulniers *et al.* (1998). Cette formulation sera utilisée dans la section 2.1.

1.2.2 Formulation basée sur les arcs

Considérons un graphe où les tâches sont représentées par des noeuds et les successions possibles entre les tâches par des arcs. Nous présentons tout d'abord un réseau avec connexions explicites (utilisé dans la section 2.2) pour ensuite agréger les arcs afin d'obtenir un réseau espace-temps (utilisé dans la section 2.3).

Réseau avec connexions explicites

Soit un réseau $G^k = (V^k, A^k)$ pour chaque dépôt $k \in K$, où V^k représente l'ensemble des noeuds et A^k l'ensemble des arcs. L'ensemble V^k contient un noeud pour chaque tâche $i \in N$ et une paire de noeuds $o(k)$ et $d(k)$ pour le dépôt (origine et destination).

L'ensemble A^k contient des arcs $(o(k),\ i)$ et $(i,\ d(k))$, $i \in N$ pour représenter les départs et les arrivées des véhicules aux dépôts. Des arcs (i, j), $i, j \in N$ relient les tâches pouvant être effectuées successivement par un même véhicule. La figure 1.1 illustre les différents types d'arcs du réseau avec connexions explicites.

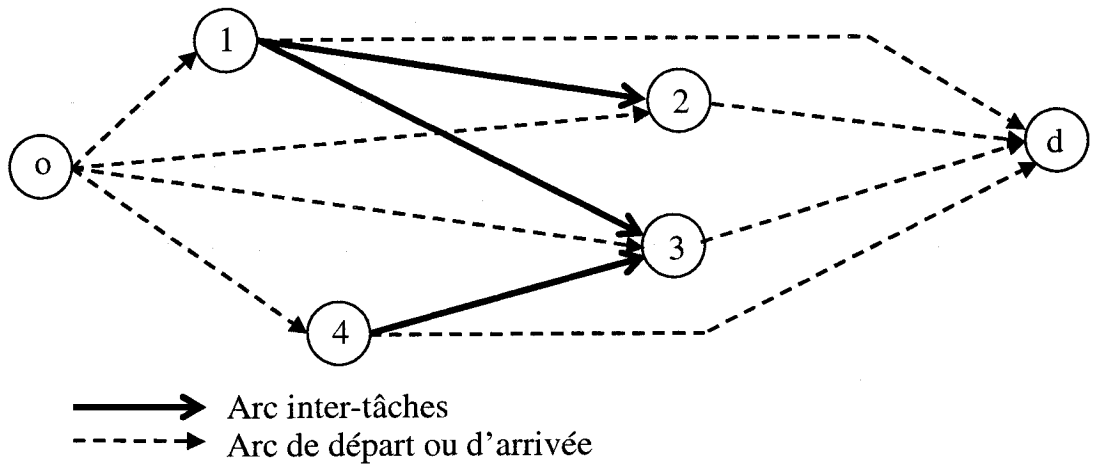

Figure 1.1 – Réseau avec connexions explicites (tâches représentées par des noeuds)

Voici une formulation plus formelle du MDVSP où X_{ij}^k est une variable binaire indiquant le flot sur l'arc (i, j) en provenance du dépôt k et c_{ij}, le coût de l'arc (i, j) :

$$Min \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij} X_{ij}^k \tag{1.5}$$

sujet à :

$$\sum_{k \in K} \sum_{j:(i,j) \in A^k} X_{ij}^k = 1, \qquad \forall i \in N \tag{1.6}$$

$$\sum_{i \in N} X_{o(k),i}^k \leq v_k, \qquad \forall k \in K \tag{1.7}$$

$$\sum_{j:(j,i) \in A^k} X_{ji}^k - \sum_{j:(i,j) \in A^k} X_{ij}^k = 0, \qquad \forall i \in N, \forall k \in K \tag{1.8}$$

$$X_{ij}^k \in \{0,1\}, \qquad \forall (i,j) \in A^k, \forall k \in K. \tag{1.9}$$

La fonction objectif (1.5) cherche à minimiser le coût total. Les contraintes (1.6) assurent que chaque tâche est exécutée une et une seule fois. Les contraintes (1.7) limitent le nombre de véhicules disponibles à chaque dépôt. Finalement, les contraintes (1.8) sont des contraintes de conservation de flot.

Réseau espace-temps

Considérons également un réseau $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{A}^k)$ pour chaque dépôt $k \in K$ avec $\mathcal{V}^k$, l'ensemble des noeuds et $\mathcal{A}^k$ l'ensemble des arcs. Pour obtenir un réseau espace-temps, les tâches doivent être représentées par des arcs et non plus par des noeuds. L'ensemble $\mathcal{V}^k$ contient cette fois un ensemble S de noeuds représentant le départ de chacune des tâches, un ensemble E de noeuds représentant l'arrivée de chacune des tâches et une paire de noeuds $o(k)$ et $d(k)$ pour le dépôt (origine et destination).

L'ensemble $\mathcal{A}^k$ contient, pour chaque tâche, un arc d'exécution (i,j) reliant son noeud de départ $i \in S$ à son noeud d'arrivée $j \in E$. Il contient également des arcs $(o(k), i)$, $i \in S$ et des arcs $(i, d(k))$, $i \in E$ pour représenter les départs et les arrivées des véhicules aux dépôts. D'autres arcs (i,j), $i \in E$, $j \in S$ relient les tâches pouvant se succéder dans une même tournée. La figure 1.2 reprend le réseau de la figure 1.1 en représentant cette fois les tâches par des arcs.

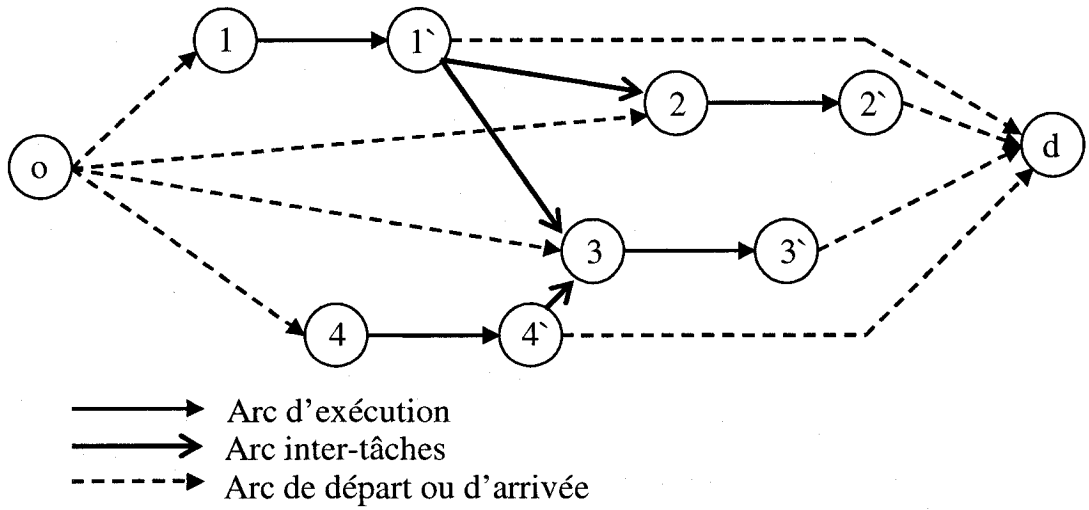

Figure 1.2 – Réseau avec connexions explicites (tâches représentées par des arcs)

On connaît, pour chaque tâche, le lieu de départ, le lieu d'arrivée, l'heure de départ et la durée. On peut donc associer à chaque noeud une heure et une station (lieu) d'événement (départ ou arrivée). Dans le réseau espace-temps, chaque station est représentée par une ligne de temps : les noeuds associés à une station sont triés selon leur ordre chronologique et reliés entre eux par des arcs d'attente. Pour représenter les départs et arrivées des véhicules, nous avons, cette fois, un arc $(o(k), i)$ et un arc $(j, d(k))$ pour chaque station, avec i, le premier noeud et j le dernier noeud de la station. Les arcs d'exécution et les arcs inter-tâches sont toujours présents.

Une économie dans le nombre d'arcs est possible grâce à l'agrégation des arcs de connexion inter-tâches (voir Kliewer *et al.*, 2005). Nous effectuons deux types d'agrégation : *premier-possible* et *dernier-possible*. L'ordre de ces agrégations est sans importance. Ils sont schématisés dans la figure 1.3.

Agrégation premier-possible

Pour chaque noeud $i \in E$ appartenant à une station a, nous conservons uniquement, pour chaque station $b \neq a$, l'arc vers la tâche compatible $j \in S$ qui débute le plus tôt.

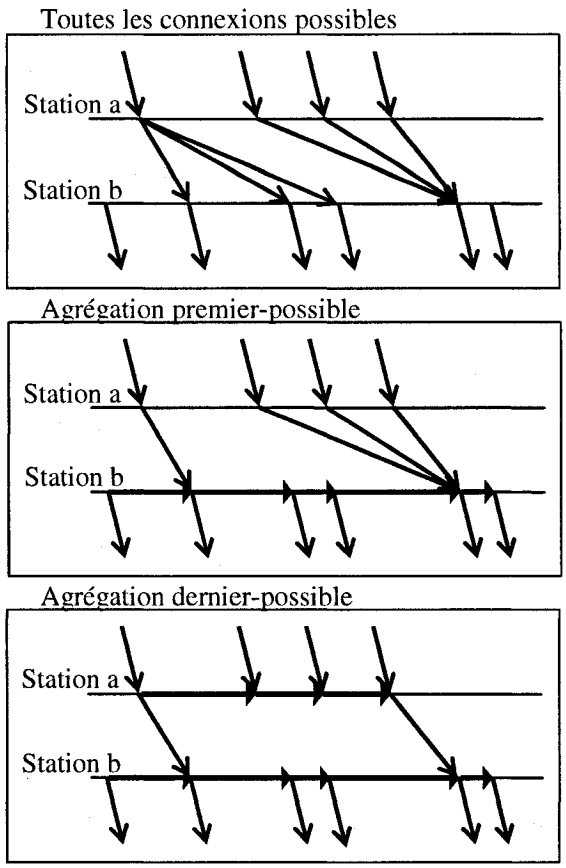

Figure 1.3 – Agrégation des connexions inter-tâches pour le réseau espace-temps

Les autres tâches débutant à la station b peuvent être effectuées directement après la tâche i par un même véhicule en utilisant l'arc conservé et les arcs d'attente de la station b.

Agrégation dernier-possible

Lorsque plusieurs arcs provenant de différents noeuds d'une même station a ont pour destination un même noeud d'une autre station b, nous conservons uniquement celui qui débute le plus tard. La tâche de la station b peut toujours être atteinte par les tâches de la station a dont l'arc a été retiré en utilisant d'abord les arcs d'attente de la station a puis l'arc conservé.

Une combinaison de l'utilisation des arcs d'attente avant et après le changement de station est possible.

Pour formuler le MDVSP en utilisant le réseau espace-temps, la notation suivante peut être utilisée :

$\mathcal{A}^k$: le nouvel ensemble des arcs agrégés ;

$\mathcal{A}_T^k$: le sous-ensemble des arcs d'exécution de $\mathcal{A}^k$;

X_{ij}^k : une variable binaire indiquant le flot sur l'arc (i,j) en provenance du dépôt k ;

c_{ij} : le coût de l'arc (i,j).

Alors le MDVSP se formule comme suit :

$$Min \sum_{k \in K} \sum_{(i,j) \in \mathcal{A}^k} c_{ij} X_{ij}^k \tag{1.10}$$

sujet à :

$$\sum_{k \in K} \sum_{(i,j) \in \mathcal{A}_T^k} X_{ij}^k = 1, \qquad \forall i \in S \tag{1.11}$$

$$\sum_{j:(j,i) \in \mathcal{A}^k} X_{ji}^k - \sum_{j:(i,j) \in \mathcal{A}^k} X_{ij}^k = 0, \qquad \forall i \in S \cup E, \forall k \in K \tag{1.12}$$

$$0 \leq X_{ij}^k \leq u_{ij}^k, \qquad \forall (i,j) \in \mathcal{A}^k, \forall k \in K \tag{1.13}$$

$$X_{ij}^k entiers, \qquad \forall (i,j) \in \mathcal{A}^k, \forall k \in K. \tag{1.14}$$

La borne supérieure sur le flot d'un arc (i,j), u_{ij}^k, est définie comme suit :

$$u_{ij}^k = \begin{cases} 1 & \text{si } (i,j) \in \mathcal{A}_T^k \\ v_k & \text{si } i = o(k) \\ \sum\limits_{k \in K} v_k & \text{sinon.} \end{cases}$$

1.3 Instances du problème

Le générateur d'instances utilisé provient de Carpaneto et al. (1989). Dans les problèmes particuliers qui nous intéressent, chaque tâche possède un point de départ et un point d'arrivée notés (x_i^d, y_i^d) et (x_i^f, y_i^f), respectivement. La distance entre les tâches i et j est $d_{ij} = \sqrt{(x_i^f - x_j^d)^2 + (y_i^f - y_j^d)^2}$ et le temps de parcours entre la fin de la tâche i et le début de la tâche j est défini par $t_{ij} = \lfloor d_{ij} \rfloor$. Chaque dépôt k possède également un couple de coordonnées (x_k, y_k) correspondant à sa localisation. Pour chaque dépôt k et chaque tâche i, il y a deux distances soit, du dépôt k au début de la tâche i, définie par $d_{ki} = \sqrt{(x_k - x_i^d)^2 + (y_k - y_i^d)^2}$ et de la fin de la tâche i au dépôt k, définie par $d_{ik} = \sqrt{(x_i^f - x_k)^2 + (y_i^f - y_k)^2}$. Notons qu'il est possible de partir et revenir au dépôt à n'importe quelle heure.

Le coût associé à une tournée correspond à la somme des coûts c_{ij} pour toute succession de tâches et/ou dépôts appartenant à la tournée. Nous utilisons un coût fixe $c_{fixe} = 10000$, soit un nombre assez grand pour s'assurer que l'objectif premier sera de minimiser le nombre de véhicules avant les coûts de déplacements. Le coût c_{ij} est défini comme suit :

$$c_{ij} = \begin{cases} 0 & \text{si } i \in K \text{ et } j \in K \\ c_{fixe}/2 + \lfloor 10 d_{ij} \rfloor & \text{si } i \in K \text{ ou } j \in K \\ \lfloor 10 d_{ij} \rfloor & \text{sinon.} \end{cases}$$

Pour générer une instance, on commence par déterminer, selon une loi uniforme, le nombre de véhicules disponibles à chaque dépôt $v_k \hookrightarrow \mathcal{U}[\lceil 3 + \frac{n}{3m} \rceil, \lfloor 3 + \frac{n}{2m} \rfloor]$ et le nombre de points de relève distincts $r \hookrightarrow \mathcal{U}[\lceil n/3 \rceil, \lfloor n/2 \rfloor]$. À chaque point de relève est affecté un couple de coordonnées $(x, y) \hookrightarrow \mathcal{U}[0, 60] \times \mathcal{U}[0, 60]$ correspondant à sa localisation géographique.

40% des tâches ne se terminent pas à l'endroit où elles ont commencé. Ces tâches, dites de type A, représentent un parcours sur une ligne d'autobus. Un point de relève pour le départ et un point de relève pour l'arrivée sont attribués aléatoirement à chacune de ces tâches. Les 60% de tâches restantes se voient attribuer un seul point de relève, à la fois pour le point de départ et d'arrivée. Ces tâches de type B représentent des allers et retours successifs sur une ligne d'autobus. Les tâches de type A sont également réparties selon la période de la journée : 15% à l'heure de pointe du matin (A_1), 15% lors de la période de pointe du soir (A_2) et 70% pour le reste de la journée (A_3). La durée de chacune des tâches est définie dans le tableau 1.1 qui résume les informations concernant les types de tâches.

Tableau 1.1 – Distribution et caractéristiques des différents types de tâches

Type	Proportion	Heure de début	Durée
A_1	$15\% \times 40\%$	$\hookrightarrow \mathcal{U}[7h00, 8h00]$	$\hookrightarrow 5 + \mathcal{U}[0, 35] + \lfloor d_{ii} \rfloor$
A_2	$15\% \times 40\%$	$\hookrightarrow \mathcal{U}[17h00, 18h00]$	$\hookrightarrow 5 + \mathcal{U}[0, 35] + \lfloor d_{ii} \rfloor$
A_3	$70\% \times 40\%$	$\hookrightarrow \mathcal{U}[8h00, 17h00]$	$\hookrightarrow 5 + \mathcal{U}[0, 35] + \lfloor d_{ii} \rfloor$
B	60%	$\hookrightarrow \mathcal{U}[5h00, 20h00]$	$\hookrightarrow 180 + \mathcal{U}[0, 120]$

Selon les auteurs, les instances ainsi générées simulent des problèmes réels en transport urbain. Ce n'est pas tout à fait le cas car il y a peu d'activités par station, le nombre de départ et d'arrivée n'est pas nécessairement équilibré et les heures d'événements sont quelconques. Ces instances modélisent plutôt les MDVSP en transport de marchandises pleine cargaison. Elles ont été retenues dans ce mémoire car elles ont souvent été utilisées dans la littérature.

Les méthodes décrites dans les prochains chapitres ont été testées sur des problèmes de 4 et 8 dépôts avec 500, 1000 et 1500 tâches.

1.4 Revue de littérature

Dans la littérature, il n'y a pas d'heuristiques récentes qui ont été développées pour résoudre le MDVSP. Les heuristiques ont surtout été développées pour ce problème avant le début des années 1990. Des méthodes exactes ont ensuite été proposées et ce, dès la fin des années 1980. Notons que plusieurs articles de synthèse ont été publiés à ce sujet dont ceux de Bodin *et al.* (1983), Odoni *et al.* (1994), Desrosiers *et al.* (1995) et Desaulniers et Hickman (2003). Dans cette section, nous proposons une brève revue sur les méthodes qui nous semblent les plus importantes.

1.4.1 Méthodes exactes

Parmi les premiers travaux, on retrouve ceux de Carpaneto *et al.* (1989), Forbes *et al.* (1994) et Bianco *et al.* (1994). Nous allons décrire ici les travaux les plus pertinents.

Ribeiro et Soumis (1994) ont proposé de formuler le MDVSP comme un problème de partitionnement d'ensemble avec contraintes supplémentaires dont la relaxation linéaire peut se résoudre par génération de colonnes. Ils montrent également que cette relaxation fournit une borne inférieure beaucoup plus serrée que les procédures précédemment utilisées pour résoudre ce problème. Les sous-problèmes sont des problèmes de plus court chemin et les décisions de branchement sont prises sur les arcs des sous-problèmes. Cette modélisation leur permet de résoudre des instances comportant jusqu'à 600 tâches et 6 dépôts.

Bianco *et al.* (1994) ont également formulé, durant la même période, le MDVSP comme un problème de partitionnement d'ensemble avec contraintes supplémentaires qu'ils résolvent à l'aide d'une méthode de séparation et évaluation progressive. Toutefois, ils réduisent d'abord le nombre de tournées admissibles selon le coût réduit des variables correspondantes au lieu de résoudre directement cette formulation.

Löbel (1998) résout à l'optimalité, du moins en ce qui concerne le nombre de véhicules, des instances de 25000 tâches et 44 dépôts grâce à une méthode de "Lagrangian pricing" associée à une méthode de génération de colonnes. Contrairement à Ribeiro et Soumis (1994), Löbel (1998) applique la génération de colonnes sur la formulation basée sur les arcs. Cette technique permet de générer des variables dont le coût réduit n'est pas négatif mais qui forment un chemin lorsque combinées avec des variables de coût réduit négatif. De plus, ils utilisent une procédure d'arrondi qui mène bien souvent à des solutions optimales.

Kliewer *et al.* (2005) ont introduit une nouvelle modélisation, présentée à la section 1.2, soit un réseau espace-temps pour réduire la taille des problèmes pratiques. Ils ont ainsi pu résoudre des instances avec 7000 tâches et 5 dépôts.

Hadjar *et al.* (2006) ont développé un algorithme qui utilise les concepts de Ribeiro et Soumis (1994) tout en permettant, de façon semblable à Bianco *et al.* (1994), d'éliminer des variables dont le coût réduit est trop élevé. Ils génèrent également des plans coupants pour resserrer la relaxation linéaire du problème. Grâce à leur technique, ils peuvent résoudre des problèmes allant jusqu'à 800 tâches et 6 dépôts.

Dans Löbel (1998) et Kliewer *et al.* (2005), les tests ont été faits sur des instances réelles, reliées aux problèmes de type horaires d'autobus urbains. Ces instances sont beaucoup plus faciles à résoudre que les instances générées aléatoirement comme dans Carpaneto *et al.* (1989). En effet, ces premières instances possèdent une structure particulière qui rend possible la résolution de problèmes de quelques milliers de tâches. La structure permet de faciliter le choix de la succession de certaines tâches car les départs et arrivées à un même lieu se suivent également dans le temps. Dans le deuxième cas, des instances de 800 tâches peuvent être résolues à l'optimalité.

1.4.2 Heuristiques

Nous allons présenter quelques-uns des travaux utilisant des heuristiques pour résoudre le MDVSP. Pour une documentation plus complète, voir Löbel (1997) et Dell'Amico *et al.* (1993). Bien que traitant d'autres problèmes, les deux articles suivants peuvent être intéressants. Cordeau *et al.* (2004) recensent quelques métaheuristiques proposées pour le problème de tournées de véhicules. Ils fournissent également des résultats numériques comparatifs. De même, Ahuja *et al.* (2002) font une étude sur les algorithmes à très large voisinage appliqués sur des problèmes NP-durs comme le voyageur de commerce.

Bodin *et al.* (1978) décrivent un algorithme d'affectations successives pour le MDVSP. Les tâches sont d'abord triées en ordre croissant selon l'heure de début pour être ensuite affectées, une à la fois, au véhicule qui minimise le coût d'insertion. Si aucune affectation n'est possible, un nouveau véhicule est utilisé, choisi parmi les dépôts ayant des véhicules disponibles et minimisant le coût d'insertion.

Un algorithme à deux phases est présenté dans Bodin *et al.* (1983). Dans la première phase, les coûts entre les dépôts et les tâches sont fixés à 0 et le MDVSP résultant (un problème simplifié) est résolu de façon exacte. Pour cela, il suffit de résoudre un problème d'affectation à coût minimum où le coût d'une succession impossible (i, j) est fixé à M, une valeur positive assez élevée. Dans la deuxième phase, les tournées déterminées dans la première phase sont réassignées aux différents dépôts. Il s'agit d'un problème de transport où les sources correspondent aux dépôts et les puits aux tournées.

Le MDVSP peut se décrire par des contraintes de couverture des tâches (chaque tâche effectuée une et une seule fois) et des contraintes de conservation de flot (chaque véhicule effectue une tournée valide). Deux types de relaxation lagrangienne peuvent

donc s'imposer : une relaxation des contraintes de couverture des tâches menant à des sous-problèmes indépendants de flot à coût minimum et une relaxation des contraintes de conservation de flot menant à la résolution de SDVSP (un seul dépôt). Le premier type est appliqué dans Kokott et Löbel (1996) alors que l'on retrouve le deuxième type de relaxation dans Kokott et Löbel (1996), Branco, Costa et Paixão (1995), Lamatsch (1992) et Mesquita et Paixão (1992). Dans tous les cas, un algorithme de sous-gradient est utilisé pour résoudre le sous-problème lagrangien.

Kokott et Löbel (1996) ont testé les deux types de relaxation lagrangienne. Lorsque les contraintes de couverture de tâches sont relaxées, le problème est décomposé en autant de problèmes de flot à coût minimum indépendants qu'il y a de dépôts. Dans l'autre cas, le sous-problème lagrangien devient un problème de flot à coût minimum avec un seul dépôt. Lamatsch (1992) obtient le même sous-problème en relaxant également les contraintes de conservation de flot. Il décrit cependant un algorithme d'étiquetage pour éliminer les conflits de véhicules sur une même tournée et trouver des solutions entières réalisables.

Mesquita et Paixão (1992), avec une formulation un peu différente, décomposent le sous-problème lagrangien en deux problèmes : un problème de tournées de véhicules sans la contrainte de passer par un dépôt et un problème où chaque tâche doit être assignée à un dépôt. Pour obtenir des solutions entières, ils échangent des bouts de tournées entre tournées de différents dépôts.

Dell'Amico *et al.* (1993) ont étudié certaines propriétés du MDVSP et les ont utilisées pour concevoir une heuristique garantissant l'utilisation du nombre minimum de véhicules possible. À chaque étape de l'algorithme, ils définissent un ensemble d'arcs interdits pour ensuite trouver un chemin réalisable (algorithme de plus court chemin) qui n'utilise aucun des arcs interdits. Ils ont également implanté quatre procédures de raffinement pour améliorer les solutions réalisables. Ils fournissent des résultats jusqu'à 10 dépôts et 1000 tâches.

Desaulniers *et al.* (1998) ont testé des approches exactes et heuristiques pour le MDVSPTW (MDVSP avec fenêtres de temps). Ils ont montré qu'une méthode de génération de colonnes imbriquée dans une procédure d'évaluation et de séparation progressive peut résoudre de grandes instances du MDVSPTW de façon exacte (300 tâches et 5 dépôts) et peut facilement être adaptée heuristiquement pour résoudre des instances encore plus grandes.

1.5 Contributions du mémoire

Les derniers travaux que nous avons trouvés sur les heuristiques pour ce problème sont ceux de Dell'Amico *et al.* (1993) et ceux présentant des méthodes basées sur la relaxation lagrangienne dans les années 1990. Dans ce mémoire, nous proposons une étude comparative impliquant les méthodologies les plus couramment utilisées de nos jours. En particulier, nous comparons les performances de trois méthodes basées sur des outils de programmation mathématique et de deux approches métaheuristiques. Il n'y a pas, à notre connaissance, de telle comparaison dans la littérature sur les problèmes de tournées de véhicules, en particulier lorsque les instances et l'environnement des tests sont les mêmes.

De plus, pour les métaheuristiques présentées dans Dereu (2004), nous avons ajouté une façon de générer une solution initiale qui permet de commencer la recherche avec le nombre minimal de véhicules et nous avons apporté des modifications mineures mais efficaces aux méthodes de recherche taboue et de recherche à voisinage large.

Nous montrons également que la réduction du nombre d'arcs du problème initial a une influence positive sur les temps de calcul pour les méthodes de génération de colonnes et de séparation et évaluation progressive sans détériorer la qualité des solutions. Finalement, nous introduisons une méthode pour tenter d'obtenir des solutions entières plus rapidement pour la méthode de génération de colonnes.

Chapitre 2 : Méthodes de résolution

Dans le mémoire de Dereu (2004), trois méthodes ont été proposées et étudiées pour résoudre le MDVSP. Les sections 2.1, 2.4 et 2.5 décrivent ces méthodes ainsi que les modifications que nous y avons apportées pour les améliorer. La section 2.1 présente une méthode de génération de colonnes heuristique. La section 2.4 décrit une méthode de recherche taboue suivie d'une méthode de recherche à voisinage large dans la section 2.5. Nous exposons également aux sections 2.2 et 2.3, deux autres algorithmes, soit une heuristique lagrangienne et une méthode de séparation et évaluation progressive heuristique, que nous trouvons pertinent de comparer avec les autres méthodes. La dernière section contient des résultats permettant de comparer la performance de ces cinq algorithmes.

2.1 Méthode de génération de colonnes heuristique

Dans cette section, nous présentons une méthode de génération de colonnes imbriquée dans une procédure de séparation et évaluation progressive utilisée pour résoudre des instances du problème décrit au chapitre 1. La méthode de génération de colonnes a été introduite par Dantzig et Wolfe (1960) et Gilmore et Gomory (1961). Nous utilisons une version heuristique de cet algorithme, similaire à celle proposée par Desaulniers *et al.* (1998) pour le MDVSP avec fenêtres de temps et coûts d'attente. La formulation retenue ici pour résoudre le MDVSP est celle présentée dans la section 1.2.1.

2.1.1 Génération de colonnes

Étant donné le très grand nombre de variables dans le modèle (1.1)-(1.4), la relaxation linéaire associée à chaque noeud de l'arbre de branchement est résolue par une méthode de génération de colonnes qui résout alternativement un problème maître restreint et un ou plusieurs sous-problèmes. Cette relaxation linéaire, aussi appelée le problème maître, s'obtient en remplaçant les contraintes (1.4) par $\theta_p^k \geq 0$.

Problème maître restreint

La méthode de génération de colonnes évite l'énumération de toutes les variables associées aux tournées réalisables en ne considérant qu'un petit sous-ensemble de ces tournées. À chaque sous-ensemble $\Omega_i^k \subseteq \Omega^k$ est associé un problème réduit, appelé problème maître restreint. À chaque itération i de la méthode de génération de colonnes, le problème maître restreint est résolu. Son rôle est de trouver la solution optimale du problème réduit (correspondant à une solution réalisable du problème maître) et de calculer les vecteurs des variables duales α_i et β_k associés aux contraintes (1.2) et (1.3), respectivement, et utilisés dans les sous-problèmes.

Selon les tests effectués par Dereu (2004), pour les instances de 500 tâches, l'algorithme primal du simplexe est utilisé pour la résolution du problème maître restreint. Pour les instances de 1000 et 1500 tâches, la méthode de barrière donne de meilleurs temps de calcul.

Sous-problèmes

Dans notre cas, les sous-problèmes sont des problèmes de plus court chemin. Pour chaque dépôt k, on cherche alors le plus court chemin de $o(k)$ vers $d(k)$ dans le réseau

avec connexions explicites G^k où les coûts des arcs sont modifiés. Le coût des arcs inter-tâches est $c_{ij} - \alpha_i$. Les arcs $(o(k), i)$ ont un coût de $c_{o(k)i} - \beta_k$ et les arcs $(i, d(k))$ un coût de $c_{id(k)} - \alpha_i$. Le coût d'un chemin p correspond alors au coût réduit de la variable θ_p correspondante.

Itérations

Pour prouver que la solution courante du problème maître restreint est aussi optimale pour le problème maître, la valeur optimale de chaque sous-problème doit être non négative. Dans le cas contraire, la solution des sous-problèmes fournit au moins une colonne à ajouter à Ω_i^k pour former Ω_{i+1}^k. Le problème maître restreint ainsi obtenu sera résolu à la prochaine itération, les variables duales seront mises à jour et utilisées pour modifier le coût des arcs dans les sous-problèmes. Le processus se poursuit de cette façon jusqu'à ce que le critère d'optimalité soit atteint.

2.1.2 Recherche d'une solution entière

Une version heuristique de cette approche est utilisée pour nos problèmes. En effet, la résolution du problème maître par génération de colonnes ne limite pas les flots sur les tournées à des valeurs entières. Les flots fractionnaires nous forcent donc à utiliser une méthode de séparation et évaluation progressive. À chaque noeud de l'arbre de branchement, le processus itératif entre le problème maître restreint et les sous-problèmes est appliqué et permet de calculer une borne inférieure sur la valeur optimale entière pour le sous-arbre engendré à ce noeud de branchement. On explore l'arbre par une stratégie de profondeur d'abord, sans retour possible vers l'arrière. L'exploration de l'arbre se termine lorsque l'on atteint une solution entière à la fin du processus de génération de colonnes (à la fin d'un noeud).

Grâce à différents tests effectués, le branchement sur les variables θ_p^k s'est avérée la méthode la plus rapide. Il s'agit de la stratégie utilisée dans Haase *et al.* (2001). On applique, à chaque noeud de l'arbre de branchement, la stratégie suivante : nous fixons à 1 toutes les variables θ_p^k dont la valeur est supérieure à un seuil pré-déterminé, soit 0,7 dans notre cas. Ces colonnes sont retirées du problème, de même que les contraintes (1.2) associées aux tâches qu'elles couvrent. Les membres de droite des contraintes (1.3) sont ajustés en conséquence.

2.1.3 Algorithme

La figure 2.1 résume le fonctionnement de l'algorithme présenté. Le rectangle en pointillé illustre un noeud de l'arbre de branchement. Dans ce noeud, la relaxation linéaire du MDVSP est résolue par une méthode de génération de colonnes (itérations entre le problème maître restreint et les sous-problèmes). La solution obtenue est optimale si elle est entière. Sinon, des décisions de branchement sont prises et sur lesquelles on ne reviendra plus. Suite à ces décisions de branchement, un nouveau noeud est exploré. Cette approche ne garantit pas la convergence de l'algorithme vers une solution entière admissible. Toutes nos instances fournissent cependant une telle solution.

2.2 Heuristique lagrangienne

Dans cette section, nous présentons une heuristique basée sur la relaxation lagrangienne pour résoudre le MDVSP. Nous remercions Dennis Huisman de l'Université Erasmus de Rotterdam qui nous a gracieusement fourni l'exécutable de son heuristique lagrangienne afin de nous permettre de comparer cette méthode avec celles que nous avons développées.

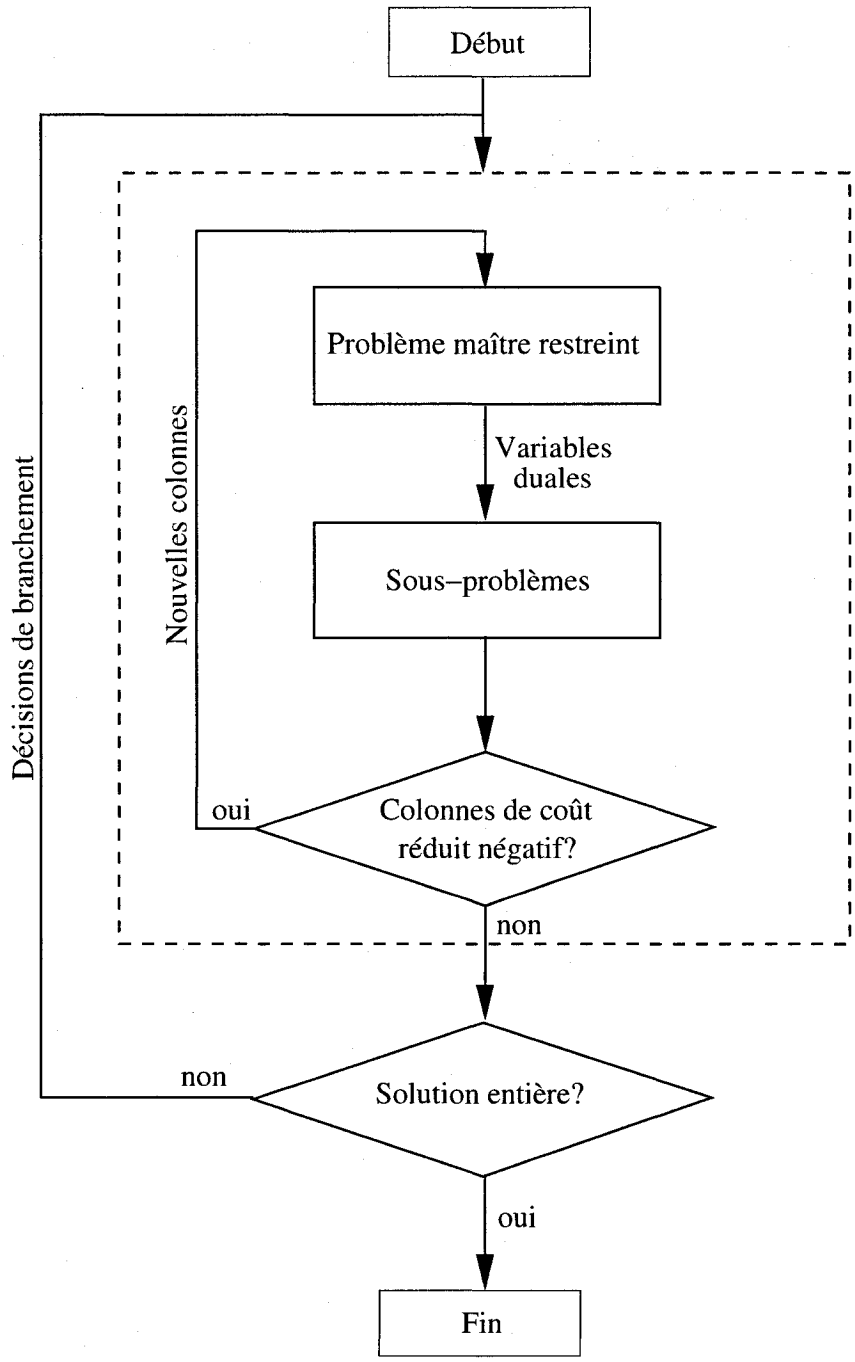

Figure 2.1 – Génération de colonnes imbriquée dans une méthode de séparation et évaluation progressive

L'approche de relaxation lagrangienne se base sur la formulation (1.5)-(1.9) à laquelle sont ajoutées les contraintes redondantes suivantes :

$$\sum_{k \in K} \sum_{j:(j,i) \in A^k} X_{ji}^k = 1, \qquad \forall i \in N. \tag{2.1}$$

Le sous-problème lagrangien est obtenu en relaxant les contraintes (1.8) et en négligeant les contraintes (1.7). Il peut alors se formuler comme suit, avec λ_i^k les multiplicateurs de Lagrange :

$$\phi(\lambda) = Min \sum_{k \in K} \sum_{(i,j) \in A^k} (c_{ij} + \lambda_j^k - \lambda_i^k) X_{ij}^k \tag{2.2}$$

sujet à :

$$\sum_{k \in K} \sum_{j:(i,j) \in A^k} X_{ij}^k = 1, \qquad \forall i \in N \tag{2.3}$$

$$\sum_{k \in K} \sum_{j:(j,i) \in A^k} X_{ji}^k = 1, \qquad \forall i \in N \tag{2.4}$$

$$X_{ij}^k \in \{0,1\}, \qquad \forall (i,j) \in A^k, \forall k \in K. \tag{2.5}$$

Les seules contraintes de ce sous-problème imposent qu'une unité de flot entre et sorte de chaque noeud. Sans les contraintes de conservation de flot, (2.2)-(2.5) définit un problème d'horaires de véhicules avec un seul dépôt (SDVSP), soit un problème de flot à coût minimum. En effet, on peut remplacer, pour chaque arc (i,j), les variables X_{ij}^k par une seule variable correspondant à la variable X_{ij}^k avec le plus petit coût réduit $\lambda_j^k - \lambda_i^k$. Les autres variables sont posées égales à 0. Nous avons donc un sous-problème lagrangien qui peut se résoudre facilement par un algorithme d'enchères comme dans Freling *et al.* (2001).

Pour un vecteur λ donné, $\phi(\lambda)$ donne une borne inférieure sur la valeur de la solution optimale. La meilleure borne inférieure est calculée grâce au problème dual lagrangien

$Max_\lambda\ \phi(\lambda)$. Une méthode de sous-gradient est utilisée pour résoudre ce problème dual.

À chaque itération, la solution du SDVSP fournit un ensemble de chemins P allant de la source vers le puits. Chaque chemin peut contenir des arcs associés à différents dépôts. En donnant la couleur k aux arcs appartenant au dépôt k, les chemins ont des arcs qui n'ont pas tous la même couleur. Pour reconstruire une solution réalisable pour le problème original, il faut donc assigner à chaque chemin une couleur unique. Pour ce faire, nous réintégrons les contraintes (1.7), précédemment négligées, en limitant à v_k le nombre de chemins avec la couleur k. Si Z_p^k est une variable binaire valant 1 si le chemin p a la couleur k et c_p^k est le coût engendré par l'assignation de la couleur k au chemin p, alors la coloration optimale des chemins peut être obtenue en résolvant le programme linéaire suivant :

$$Min \sum_{k\in K}\sum_{p\in P} c_p^k Z_p^k \tag{2.6}$$

sujet à :

$$\sum_{k\in K} Z_p^k = 1, \qquad \forall p \in P \tag{2.7}$$

$$\sum_{p\in P} Z_p^k \leq v_k, \qquad \forall k \in K \tag{2.8}$$

$$Z_p^k \in \{0,1\}, \qquad \forall p \in P, \forall k \in K. \tag{2.9}$$

Ce problème correspond à un problème de transport qui peut être résolu en temps polynomial par la méthode hongroise (voir Ahuja *et al.*, 1993) et fournit une borne supérieure pour le MDVSP. Une telle borne est calculée pour chaque itération où la borne inférieure est améliorée. Contrairement à la méthode de génération de colonnes exposée dans la section précédente, l'approche de relaxation lagrangienne permet de générer des solutions réalisables tout au long du processus de résolution.

L'algorithme 2.1 donne les détails de l'algorithme utilisé où c est utilisé comme compteur. L'algorithme est initialisé avec des multiplicateurs égaux à 0. Le sous-problème lagrangien est résolu et un sous-gradient est calculé à l'étape 1. À l'étape 2, la borne supérieure du problème est mise à jour si elle s'est améliorée. Dans l'étape 3, le vecteur du sous-gradient est utilisé pour la mise à jour des multiplicateurs avec un pas déterminé par la différence entre les bornes supérieures et inférieures, la norme du sous-gradient et un paramètre α. À l'étape 4, α est mis à jour pour assurer la convergence de l'algorithme. Finalement, les critères d'arrêt sont un sous-gradient nul, une borne inférieure égale à une borne supérieure, α très petit ou un nombre maximum d'itérations. Les valeurs suivantes de paramètres ont été utilisés : $\alpha^0 = 1,0$, $\gamma = 10$, $\epsilon = 0,000001$ et $\eta_{max} = 10000$. Ils ont été ajustés à partir de résultats de tests préliminaires.

2.3 Algorithme de séparation et évaluation progressive heuristique

Le programme linéaire en nombres entiers (1.10)-(1.14), basé sur le réseau espace-temps décrit à la section 1.2.2 peut être résolu par un algorithme de séparation et évaluation progressive heuristique (SEP). Pour ce faire, nous avons utilisé CPLEX, version 9.0.1. Suite à des tests préliminaires, nous avons fixé l'algorithme pour résoudre la relaxation linéaire avec la méthode de barrière et nous avons demandé au logiciel de s'arrêter dès qu'une solution entière était trouvée. L'option *MIP emphasis* de CPLEX a été réglé à *feasibility and optimality*, soit de mettre l'emphase de recherche à la fois sur une solution réalisable et optimale. Ces paramètres sont ceux ayant fournis les meilleurs temps de résolution. Les autres paramètres sont ceux par défaut.

Algorithme 2.1 Heuristique lagrangienne

Étape 0 : **Initialisation**
Poser $\eta_{max} := 10000$, $\alpha^0 = 1,0$, $\gamma = 10$, $\epsilon = 0,000001$
Poser $UB := \infty$, $LB := -\infty$, $c := 0$
Commencer avec les multiplicateurs initiaux $\lambda^0 := 0$

Étape 1 : **Résolution du sous-problème lagrangien**
Résoudre le sous-problème lagrangien (2.2)-(2.5) pour obtenir une solution x^c et une borne inférieure $\phi(\lambda^c)$
Calculer le sous-gradient $y_i^{k,c} := \sum_{j:(j,i)\in A^k} X_{ji}^{k,c} - \sum_{j:(i,j)\in A^k} X_{ij}^{k,c}$

Étape 2 : **Calcul de la borne supérieure**
Calculer la borne supérieure UB^c
si $UB^c \leq UB$ **alors** $UB := UB^c$

Étape 3 : **Calcul des multiplicateurs de Lagrange**
Calculer $\lambda_i^{k,c+1} := \lambda_i^{k,c} + \alpha^c \frac{UB - \phi(\lambda^c)}{\sum_{k\in K}\sum_{i\in N}(y_i^{k,c})^2} y_i^{k,c}$

Étape 4 : **Mise à jour des paramètres**
si $\phi(\lambda^c) > LB$ **alors** $m := 0$ et $LB := \phi(\lambda^c)$
sinon $m := m + 1$
si $m = \gamma$ **alors** $\alpha^{c+1} := \alpha^c/2$
sinon $\alpha^{c+1} := \alpha^c$

Étape 5 : **Critères d'arrêt**
si $UB = \phi(\lambda^c)$, $\sum_{k\in K}\sum_{i\in N}(y_i^{k,c})^2 \leq \epsilon$, $\alpha^c \leq \epsilon$ ou $c \geq \eta_{max}$ **alors** STOP
sinon Retourner à l'étape 1

2.4 Algorithme de recherche taboue

La recherche taboue est une métaheuristique qui a été proposée en 1986 par Glover. Cette méthode s'est avérée assez efficace pour résoudre des problèmes NP-difficiles en obtenant, en des temps raisonnables, des solutions de bonne qualité. L'algorithme que nous avons utilisé pour résoudre le problème du MDVSP est basé sur celui développé par Cordeau *et al.* (2001) pour les problèmes de tournées de véhicules avec dépôts multiples.

La recherche taboue est une méthode itérative de recherche locale. À chaque itération, l'algorithme explore l'espace des solutions en se déplaçant de la solution courante s à la meilleure solution appartenant à un voisinage $N(s)$ de s. De plus, s peut être non réalisable. Dans notre cas, nous permettons les violations sur l'heure de début des tâches uniquement, c'est-à-dire qu'une tâche peut être effectuée en retard. La solution peut se détériorer d'une itération à l'autre pour permettre de se sortir des minima locaux. De plus, pour éviter de revenir sur des solutions déjà visitées, les mouvements sont restreints par la présence d'une liste taboue, mise à jour à chaque itération. En général, les mouvements tabous le restent durant un certain nombre d'itérations mais peuvent être retirés de la liste s'ils satisfont un critère d'aspiration. Le critère d'aspiration que nous avons utilisé est d'autoriser un mouvement tabou s'il permet d'améliorer le coût de la meilleure solution trouvée.

L'algorithme 2.2 décrit la recherche taboue avec s la solution courante, s^* la meilleure solution trouvée et f^* la valeur de l'objectif en s^* (voir section 2.4.1). On commence par trouver une solution initiale réalisable de la façon décrite dans la section 2.4.3 et tant qu'aucun critère d'arrêt n'est satisfait, on se déplace vers la meilleure solution admissible appartenant au voisinage $N(s)$ de s. Par admissible, on entend une solution non taboue ou autorisée par aspiration. Cet ensemble de solutions admissibles est noté $N'(s) \in N(s)$.

Algorithme 2.2 Recherche taboue

1: Construction de la solution initiale s
2: $s^* \leftarrow s$
3: $f^* \leftarrow f(s)$
4: **tant que** *aucun critère d'arrêt n'est satisfait* **effectuer**
5: $s \leftarrow \text{argmin}_{s' \in N'(s)}[f(s')]$
6: **si** s *est réalisable* **et** $f(s) < f^*$ **alors**
7: $s^* \leftarrow s$
8: $f^* \leftarrow f(s)$
9: Mise à jour de la liste taboue

Parmi les critères d'arrêt les plus fréquemment utilisés dans cette méthode, on trouve un nombre fixe d'itérations (utilisé pour nos tests), un temps CPU fixe ou un nombre d'itérations sans amélioration.

2.4.1 Détails de l'algorithme de Cordeau *et al.*

Dans l'algorithme 2.2, la fonction $f(s)$ est égale à $c(s)$, le coût de la solution s, uniquement si s est réalisable. Sinon, une pénalité est ajoutée pour augmenter la valeur de $f(s)$ de façon à défavoriser les violations sur les temps de départ, c'est-à dire, défavoriser les retards sur le commencement des tâches. Dans ce cas, $f(s) = c(s) + \gamma w(s)$ avec $w(s)$ la somme des retards aux commencements des tâches et γ un paramètre positif ajusté dynamiquement.

Plus γ est élevé, moins grandes seront les chances de choisir une solution non réalisable. Au contraire, plus γ est faible, plus la solution sera de faible coût, sans pour autant être réalisable. Tout au long de l'algorithme, ce paramètre sera ajusté en fonction de la solution à l'itération courante. Si elle est réalisable, on diminuera la valeur de γ à $\gamma = \gamma/(1+\delta)$, avec δ un paramètre positif fixé à $0,5$, sinon on l'augmentera à $\gamma = \gamma(1+\delta)$.

Un processus de diversification est ajouté pour éviter que la recherche soit restreinte à un petit espace des solutions. Une fonction de pénalité p est ajoutée à f lorsque le coût de $s' \in N(s)$ est supérieur à celui de la solution courante s. Autrement dit, si $f(s') = c(s') + \gamma w(s) > f(s)$,

$$f(s') = c(s') + \gamma w(s') + p(s')$$

où

$$p(s') = \lambda c(s') \sqrt{nm} \sum_{k \in K} \sum_{i \in N} \alpha_{ik}^{s'} \rho_{ik}$$

avec

ρ_{ik} : le nombre de fois où la tâche i a été affectée au véhicule k ;

$\alpha_{ik}^{s'}$: un paramètre qui vaut 1 si, dans la solution s', la tâche i appartient à la tournée k, 0 sinon ;

λ : un paramètre aléatoire entre 0 et 1 pour contrôler l'intensité de la diversification.

2.4.2 Voisinage d'une solution et liste taboue

Le voisinage d'une solution s est défini par l'ensemble des solutions générées par le déplacement d'une tâche i d'un chemin k vers un chemin k' ou par l'échange de deux tâches appartenant à des tournées différentes. L'échange de tâches a été ajouté à la méthode implémentée par Dereu (2004). Si la nouvelle solution s' a été choisie dans le voisinage de la solution s en retirant la tâche i du chemin k, l'opération consistant à réinsérer cette tâche dans le chemin k sera interdite durant un certain nombre d'itérations et correspondra donc à un mouvement de la liste taboue. Le nombre d'itérations durant lesquelles un mouvement reste tabou suit une loi aléatoire uniforme sur $[0, L]$, où L est un paramètre permettant de choisir la longueur maximale de la liste taboue.

Suite à des tests préliminaires effectués par Dereu (2004), une bonne approximation pour L est fournie par la formule suivante avec t, une approximation sur le nombre de tournées présentes dans la solution :

$$L = \sqrt{n \times t}$$

Les tests de Dereu se limitant aux instances d'au plus 1000 tâches, nous avons choisi de prendre la même valeur de L pour les instances de 1500 tâches que celle retenue pour celles de 1000 tâches. La valeur de L a été fixée à 234 pour les instances de 500 tâches et 456 pour celles de 1000 et 1500 tâches.

2.4.3 Construction de la solution initiale

Algorithme 2.3 Construction de la solution initiale

1: **pour tout** $i \in N$ **effectuer**
2: $c_{di} \leftarrow \min_{k \in K} c_{ki}$
3: $c_{id} \leftarrow \min_{k \in K} c_{ik}$
4: Résolution avec CPLEX du SDVSP
5: // Assigner les tournées aux dépôts
6: **pour tout** $p \in$ *tournées de la solution du SDVSP* **effectuer**
7: $k_p \leftarrow \text{argmin}_{k \in K}(c_{ki_p^1} + c_{i_p^n k})$ tel que k contient au moins 1 véhicule disponible
8: Assigner la tournée $p=(i_p^1,...,i_p^l)$ au dépôt k_p
9: Réduire de 1 le nombre de véhicules disponibles au dépôt k_p

Dans Dereu (2004), plusieurs méthodes pour construire la solution initiale ont été présentées. Nous avons cependant implémenté une nouvelle façon de la construire dans la méthode de recherche taboue et dans la méthode de recherche à voisinage large (voir section 2.5). Nous utilisons une variante de la méthode introduite dans les travaux de Bodin *et al.* (1983) et utilisés dans Oukil (2006) et Marin (2005). Elle est plus efficace que celles proposées par Dereu (2004) car elle permet d'obtenir immédiatement le nombre minimum de véhicules, impliquant donc une diminution

importante du coût de départ. Pour ce faire, on a généré un problème de flot à coût minimum en transformant le problème de MDVSP en un problème à un seul dépôt (SDVSP pour Single Depot Vehicle Scheduling Problem). Le coût de déplacement du dépôt d, maintenant unique, vers chaque tâche a été calculé de la façon suivante : $c_{di} = \min_{k \in K} c_{ki}$. De la même façon, le coût d'un déplacement de la tâche i vers le puits devient $c_{id} = \min_{k \in K} c_{ik}$. Pour nos tests, le SDVSP est résolu par la version de l'algorithme du simplexe pour réseau de CPLEX. Une fois le SDVSP résolu, on doit assigner les chemins obtenus aux dépôts d'origine. L'algorithme 2.3 décrit la façon dont la solution initiale est trouvée. Dans l'algorithme, la notation i_p^1 correspond à la première tâche de la tournée p et i_p^l à la dernière tâche de cette même tournée.

2.4.4 Changement de dépôt des tournées

Nos premiers résultats ont montré que la méthode a des difficultés à sortir des minima locaux. Ceci s'explique en partie par le fait que notre voisinage ne modifie pas le nombre de véhicules utilisé dans chaque dépôt (fixé par la solution initiale). Nous avons donc implémenté une méthode pour réaffecter les tournées à des véhicules appartenant à d'autres dépôts et ce, de deux façons. Tout d'abord, lorsque la dernière tâche i_p^l (respectivement la première tâche i_p^1) d'une tournée p d'un véhicule appartenant au dépôt k est déplacée, il peut devenir avantageux de réassigner cette tournée à un véhicule d'un autre dépôt k' si $c_{k'i_p^1} + c_{i_p^{l-1}k'} < c_{ki_p^1} + c_{i_p^{l-1}k}$, avec i_p^{l-1} la nouvelle dernière tâche de la tournée p (respectivement $c_{k'i_p^2} + c_{i_p^l k'} < c_{ki_p^2} + c_{i_p^l k}$, avec i_p^2 la nouvelle première tâche de la tournée). De plus, si une tâche est insérée en début ou en fin de tournée, un calcul similaire est effectué pour cette tournée. Nous avons également forcé, pour permettre de varier l'espace des solutions, la réaffectation d'une tournée choisie aléatoirement à toutes les 150 itérations.

2.5 Recherche à voisinage large

La recherche à voisinage large (RVL) est une métaheuristique réoptimisant, à chaque itération, une partie de la solution courante. Dans notre cas, le principe consiste à choisir, à chaque itération, V tournées qui seront réoptimisées, c'est-à-dire que le sous-ensemble des tâches incluses dans ces tournées forme un nouveau MDVSP qui est résolu par la méthode de génération de colonnes présentée à la section 2.1.

Si le nombre de tournées à réoptimiser est assez petit, la résolution par génération de colonnes fournira rapidement une très bonne solution. Différents tests ont permis de constater que $V = 30$ donne de meilleurs résultats pour les instances de 500 et 1000 tâches, alors que $V = 40$ est préférable lorsque le nombre de tâches grimpe à 1500.

Un autre avantage de cette méthode est que l'on aura une solution entière réalisable à chaque itération. En effet, la solution initiale construite à partir du SDVSP (voir section 2.4.3) est réalisable. De plus, la méthode de génération de colonnes nous fournit des solutions réalisables pour chaque ensemble de tournées réoptimisées. L'algorithme 2.4 décrit cette méthode.

2.5.1 Choix des tournées à réoptimiser

Quatre stratégies ont été implantées par Dereu (2004) pour la sélection des tournées à réoptimiser. Deux de ces stratégies, soit le choix des tournées les moins chargées et le choix de la moins chargée ne sont plus utiles à cause de la nouvelle façon de trouver une solution intiale. En effet, ces stratégies avaient pour but d'aider à réduire le nombre de véhicules dans la solution, ce qui n'est plus nécessaire. Il en reste donc deux que nous avons conservées ainsi qu'une troisième que nous avons ajoutée. Plusieurs tests nous ont permis de constater que ces trois stratégies ensemble donnaient de meilleurs résultats que toute autre combinaison.

Algorithme 2.4 Recherche à voisinage large

1: Initialiser le poids des stratégies à 1
2: Construction de la solution initiale s
3: $s^* \leftarrow s$
4: $c^* \leftarrow c(s)$
5: **tant que** *aucun critère d'arrêt n'est satisfait* **effectuer**
6: Choisir une stratégie de sélection de tournées
7: Appliquer la stratégie pour trouver les V tournées à réoptimiser
8: Réoptimiser les V tournées par génération de colonnes
9: Mettre à jour la solution courante s
10: **si** $c(s) < c^*$ **alors**
11: $s^* \leftarrow s$
12: $c^* \leftarrow c(s)$
13: Mettre à jour la liste taboue
14: Mettre à jour le poids de la stratégie choisie

Ces trois stratégies sont décrites dans les paragraphes qui suivent. Le choix de la stratégie à chaque itération est inspirée de la formulation utilisée par Ropke et Pisinger (2004). À chaque itération, chaque stratégie a un poids $w_i > 0$ et une probabilité $p_i = \frac{w_i}{\sum_i w_i}$ d'être choisie. Après chaque itération, le poids de la stratégie choisie est mis à jour suivant les améliorations que cette stratégie a permises. En notant par s_{prec}, la solution avant l'application de la stratégie, le nouveau poids devient $w_i = \rho w_i + (1 - \rho)(c(s_{prec}) - c(s))$ avec $\rho \in [0, 1[$. Il correspond donc à une moyenne pondérée des améliorations et permet de donner plus de poids à une stratégie plus efficace.

La première tournée choisie par chaque stratégie est ajoutée à une liste taboue et le reste durant 20 itérations. Cette technique permet de varier le plus possible le choix des tournées à réoptimiser. Voici maintenant les trois stratégies.

Tournées aléatoires

La première tournée est choisie au hasard parmi les tournées non taboues. Ensuite, $V - 1$ tournées sont choisies aléatoirement parmi toutes les tournées restantes (ces tournées peuvent être taboues).

Tournées proches

De la même façon, une première tournée p_1 est choisie parmi les tournées non taboues. Nous essayons ensuite de sélectionner des tournées proches géographiquement et temporellement de cette tournée. Pour ce faire, nous calculons une mesure de proximité pour chaque tournée $p \neq p_1$ de valeur $min_{i \in p_1, j \in p}(\alpha c_{ij} + t_{ij}, \alpha c_{ji} + t_{ji})$ avec $t_{ij} = \infty$ si la tâche j ne peut succéder à la tâche i et α un coefficient de pondération dont la valeur a été fixée à 10 lors de nos tests.

Nous choisissons alors les V tournées avec la mesure de proximité de valeur minimum.

Tournées les moins fréquentes

À chaque itération, nous mettons à jour le nombre de fois que chaque tournée a été choisie pour être réoptimisée. Dans cette troisième stratégie, nous choisissons les V tournées les moins fréquemment sélectionnées.

2.6 Résultats

Dans cette section, nous présentons les tests que nous avons effectués pour comparer les cinq méthodes présentées dans ce chapitre et les résultats obtenus. Les tests ont

été faits sur une station de travail Intel Xeon 2.66GHz avec 1GB. Les résultats qui sont présentés ici ont été obtenus sur des problèmes de 4 et 8 dépôts avec 500, 1000 et 1500 tâches. Dans chacun des cas, les résultats constituent des moyennes sur 5 instances de la même taille. Les résultats sont présentés à l'aide de courbes dans les figures 2.2 - 2.7.

Le logiciel GENCOL 4.5 jumelé à CPLEX 9.0.1 a été utilisé pour la méthode de génération de colonnes. Les différents points de cette courbe correspondent chacun à une résolution distincte en arrêtant prématurément la résolution du problème maître. Pour ce faire, on impose d'arrêter les itérations dès que la valeur de l'objectif n'a pas diminué d'au moins Q durant I itérations. I vaut 2 ou 5 et Q varie de 0 à 7000 pour 500 tâches, de 0 à 200000 pour 1000 tâches et de 0 à 500000 pour 1500 tâches.

Pour l'heuristique lagrangienne, la recherche à voisinage large et la recherche taboue, la courbe est constituée des valeurs de l'objectif (et des temps de calcul correspondant) pour chaque itération où cette valeur s'est améliorée. Étant donné que le critère d'arrêt de la recherche taboue et la RVL est un nombre fixe d'itérations, nous avons choisi ce nombre pour obtenir des temps de calcul similaires aux temps maximums obtenus par génération de colonnes. En effet, nous avons conclu qu'il était inutile de chercher plus longtemps des solutions avec ces deux méthodes étant donné l'efficacité de cette dernière.

Les instances de 500 tâches avec 4 dépôts sont les seules dont les temps de résolution par l'algorithme de SEP sont assez courts pour permettre de visualiser sur la figure la valeur trouvée. Pour les autres tailles, le temps est beaucoup trop long pour l'échelle des graphiques. Nous donnons donc ces valeurs dans les tableaux 2.1 et 2.2, de même que les résultats numériques des autres méthodes. Les meilleurs résultats (valeur de l'objectif) sont affichés en caractères gras. Dans certains cas, la méthode de SEP a donné un meilleur résultat que la méthode de génération de colonnes mais le temps

étant très long et la différence dans la valeur de la solution très petite, le résultat de cette dernière a également été mis en évidence.

Tableau 2.1 – Comparaison numérique des heuristiques pour 4 dépôts

	500 tâches		1000 tâches		1500 tâches	
	temps	valeur	temps	valeur	temps	valeur
SEP	81	1278181,6	**1287**	**2478545,6**	**4149**	**3602758**
RL	85	1279816,9	700	2483082,4	2300	3610871,4
GC	**77**	**1278107,6**	**651**	**2478739**	**2203**	**3603044**
LNS	85	1279275,7	700	2480065,3	2300	3605551,3
Taboue	85	1284342,8	700	2488500,3	2300	3618857,8

Tableau 2.2 – Comparaison numérique des heuristiques pour 8 dépôts

	500 tâches		1000 tâches		1500 tâches	
	temps	valeur	temps	valeur	temps	valeur
SEP	612	1285640,2	**6207**	**2495918,4**	-	-
RL	125	1287924,1	900	2501648,9	3200	3637341,8
GC	**119**	**1285575,4**	**857**	**2496005,6**	**3085**	**3625731,8**
LNS	125	1286820,9	900	2498458,9	3200	3629288,8
Taboue	125	1293769,6	900	2514100,8	3200	3650643

Nous pouvons remarquer dans les figures 2.2 - 2.7 que le comportement des différentes approches (sauf la SEP) est très semblable pour toutes les tailles de problèmes. De plus, le nombre minimum de véhicules est atteint très rapidement par les quatre approches (le coût d'un véhicule supplémentaire donnerait un bond de 10000 dans la valeur de l'objectif). En fait, grâce à la méthode d'initialisation présentée à la section 2.4, la recherche taboue et la RVL commencent dès l'initialisation avec le nombre minimum de véhicules. Par contre, la méthode taboue plafonne rapidement, alors que la RVL permet d'améliorer considérablement les solutions initiales.

Le dernier tableau (2.3) de ce chapitre présente les différences, en pourcentage, entre la valeur de la solution trouvée par génération de colonnes et la valeur de la relaxation linéaire pour nos tests, lorsque $Q = 0$. Le coût fixe n'est pas considéré dans ce calcul.

Tableau 2.3 – GAP en pourcentage pour la méthode de génération de colonnes

	500 tâches	1000 tâches	1500 tâches
4 dépôts	0,1681	0,3499	0,4150
8 dépôts	0,5443	0,6671	0,8545

Dans le chapitre suivant, nous allons tenter d'obtenir des solutions plus rapidement pour la méthode de génération de colonnes et la méthode de SEP.

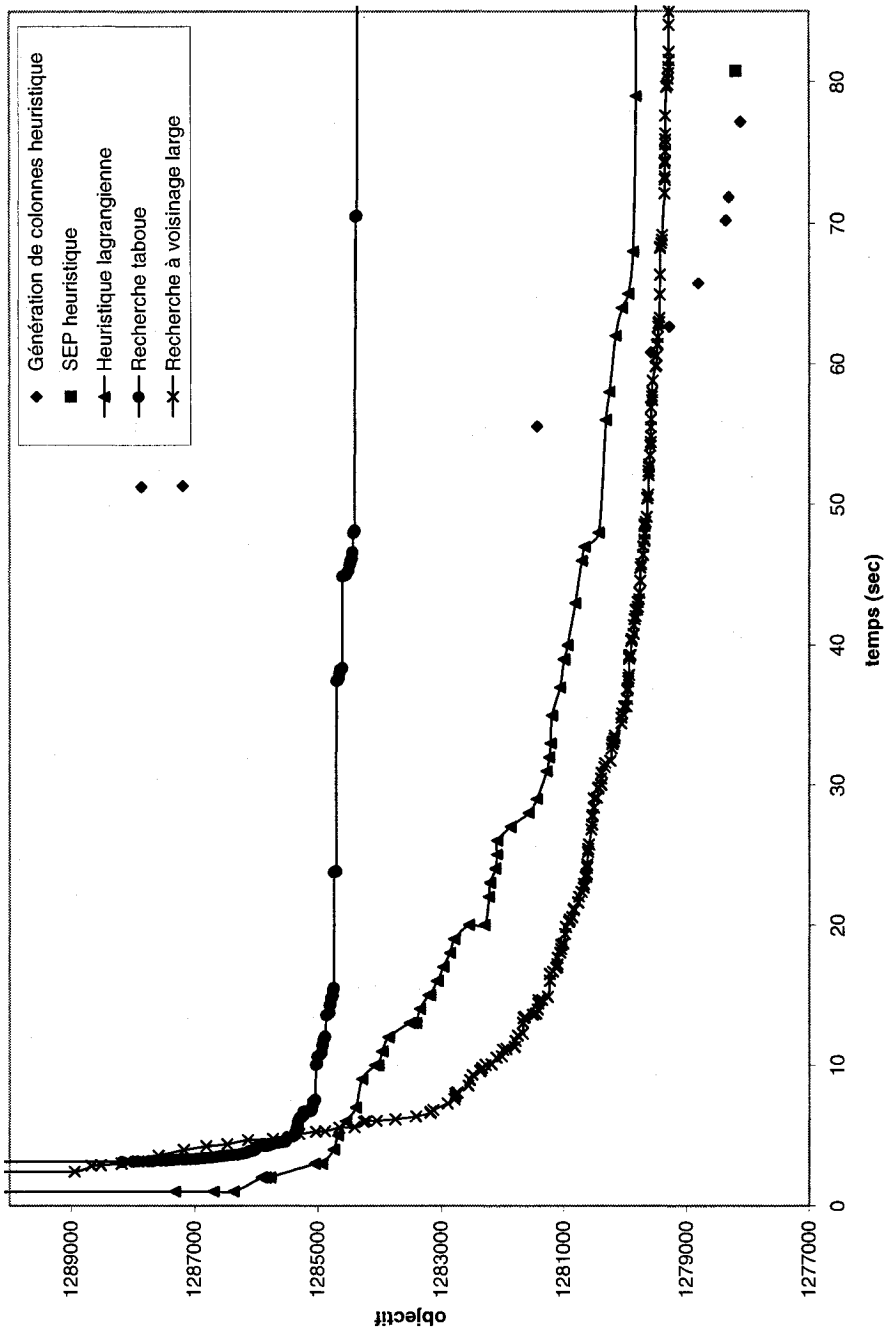

Figure 2.2 – Comparaison des heuristiques pour 500 tâches 4 dépôts

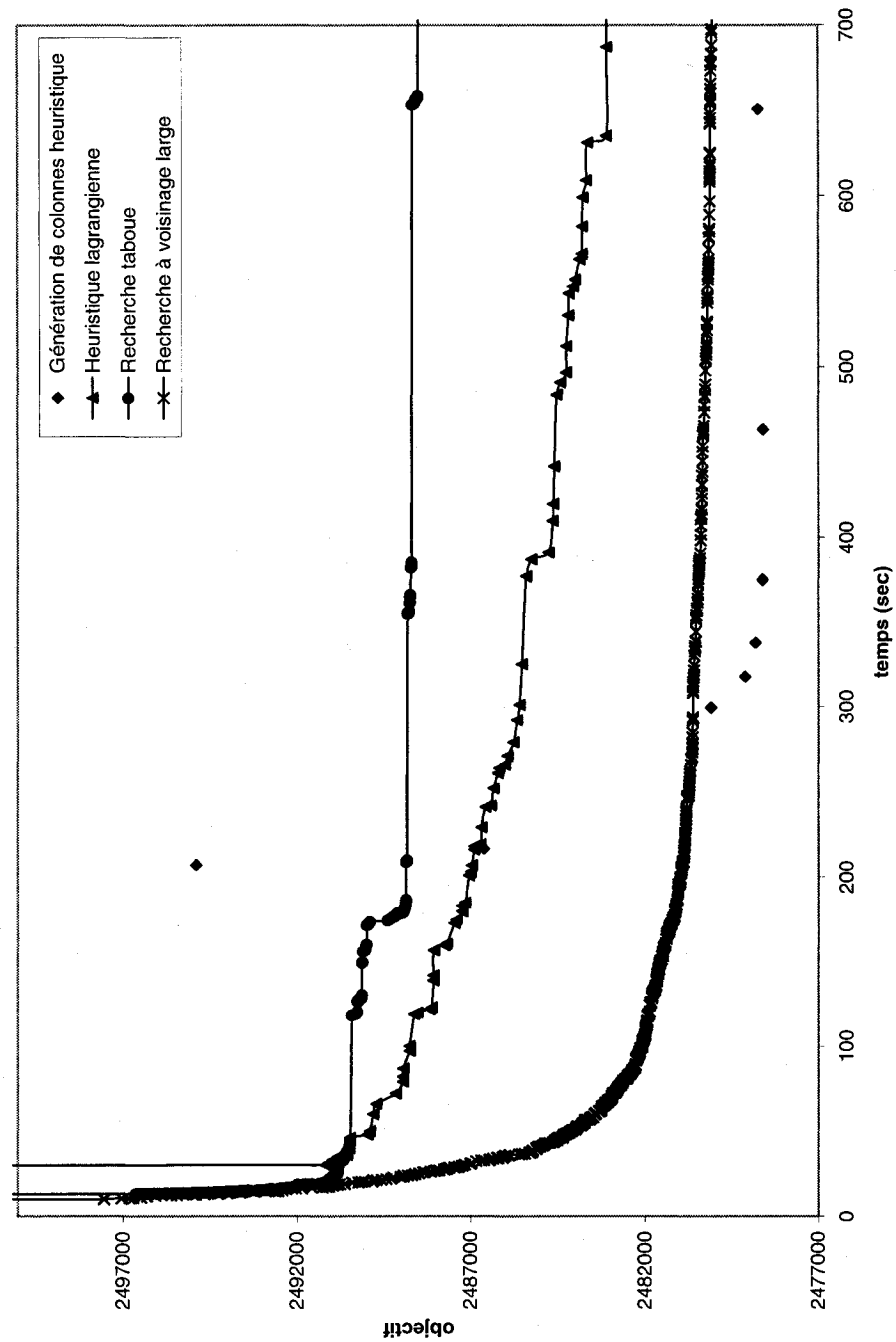

Figure 2.3 – Comparaison des heuristiques pour 1000 tâches 4 dépôts

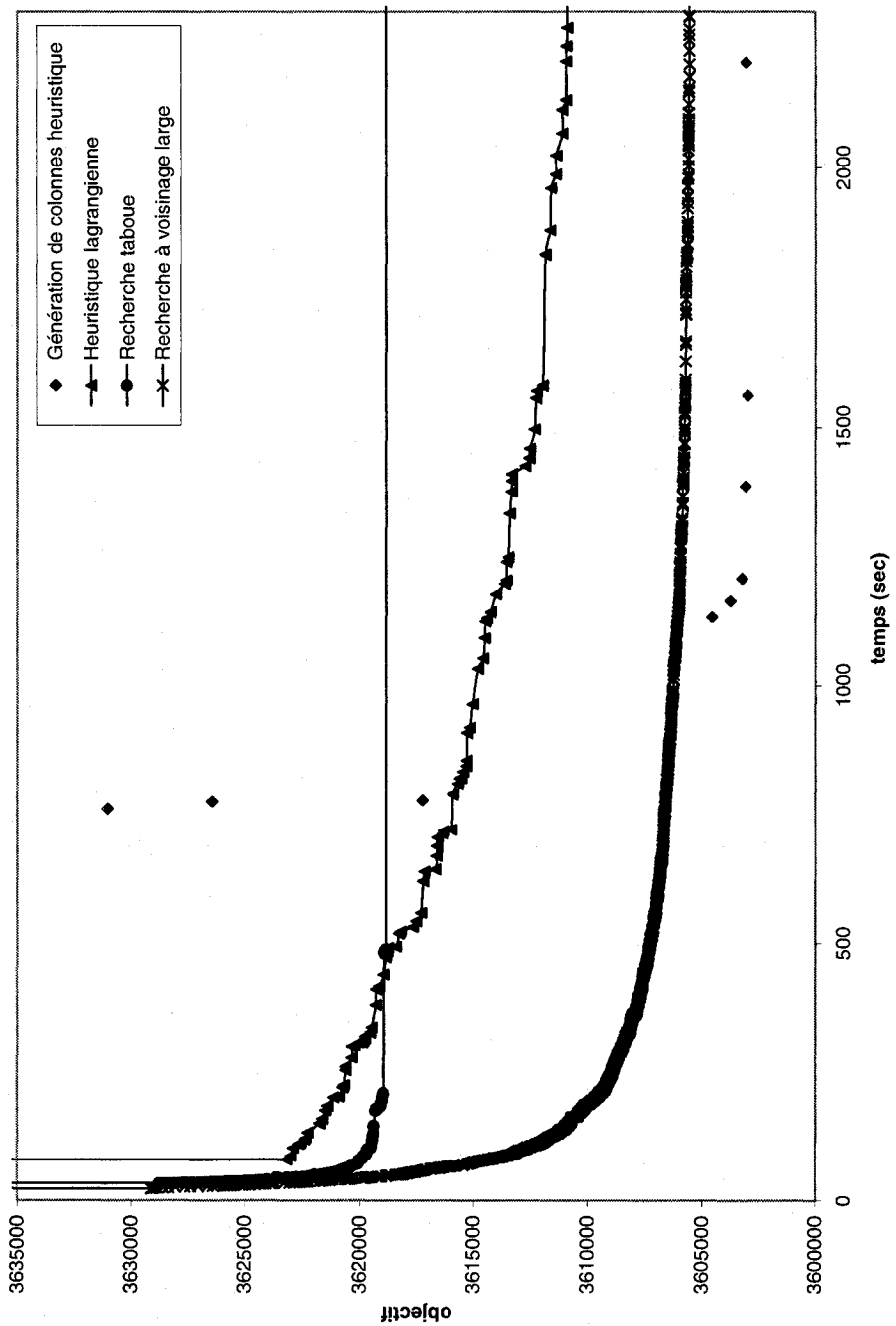

Figure 2.4 – Comparaison des heuristiques pour 1500 tâches 4 dépôts

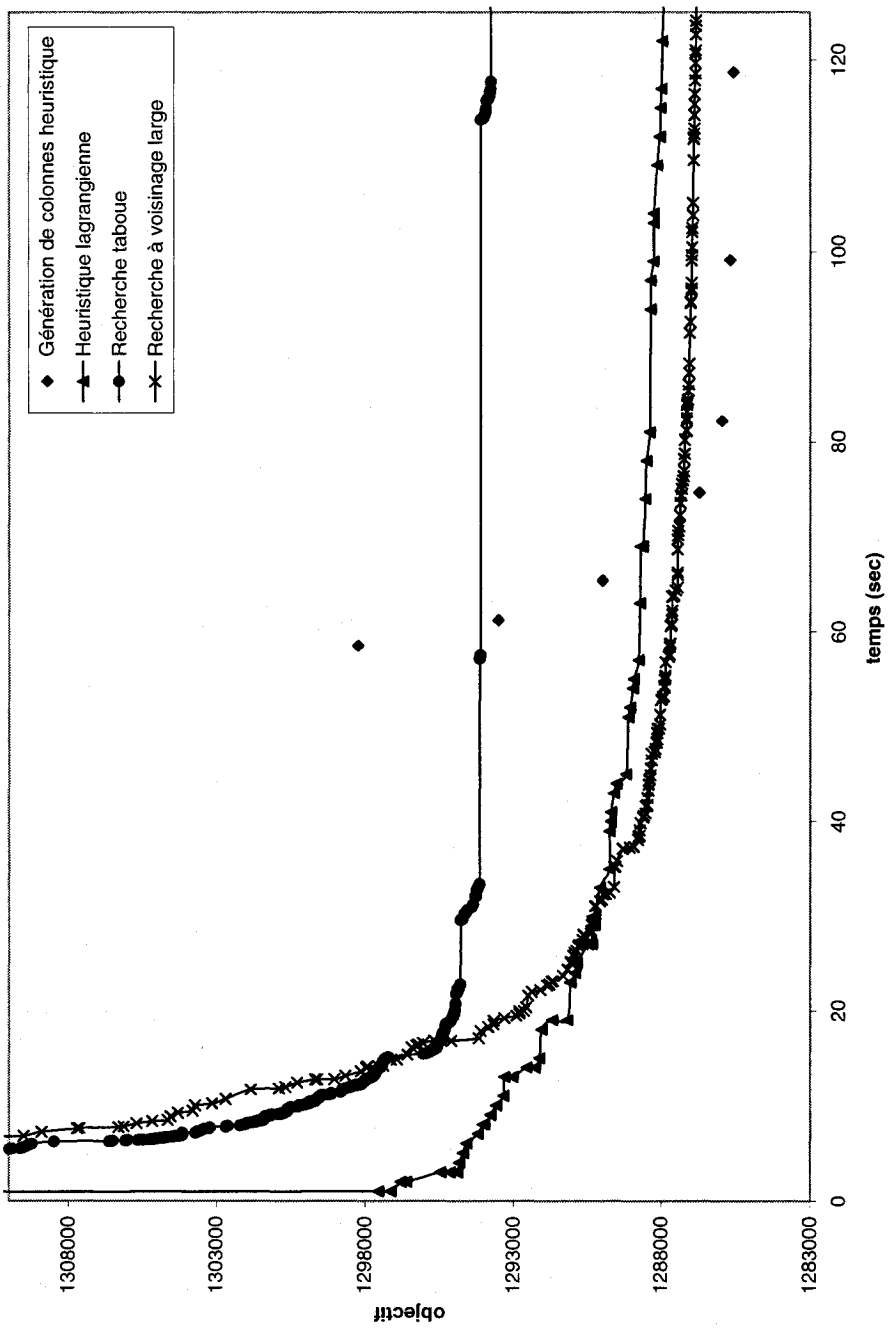

Figure 2.5 – Comparaison des heuristiques pour 500 tâches 8 dépôts

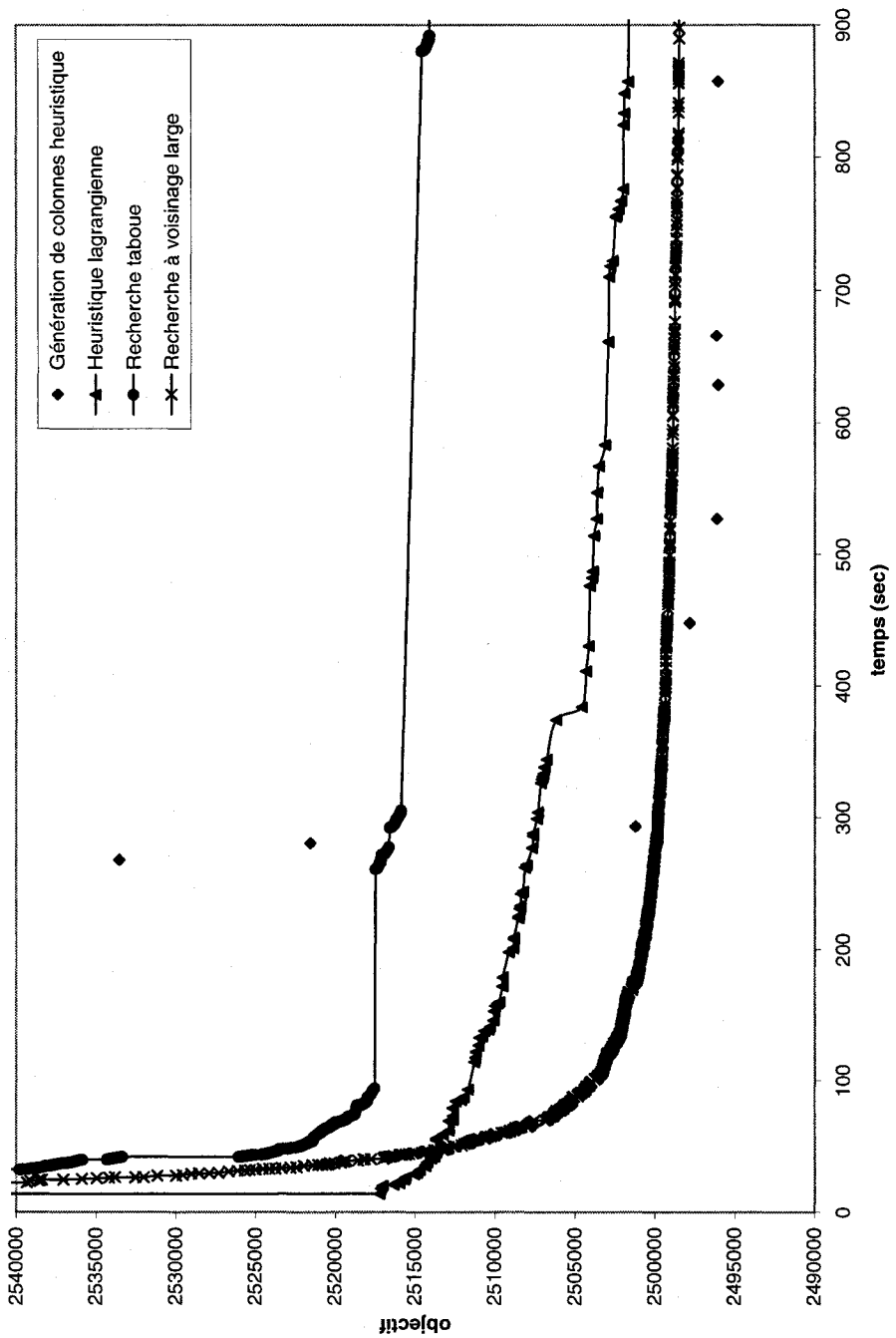

Figure 2.6 – Comparaison des heuristiques pour 1000 tâches 8 dépôts

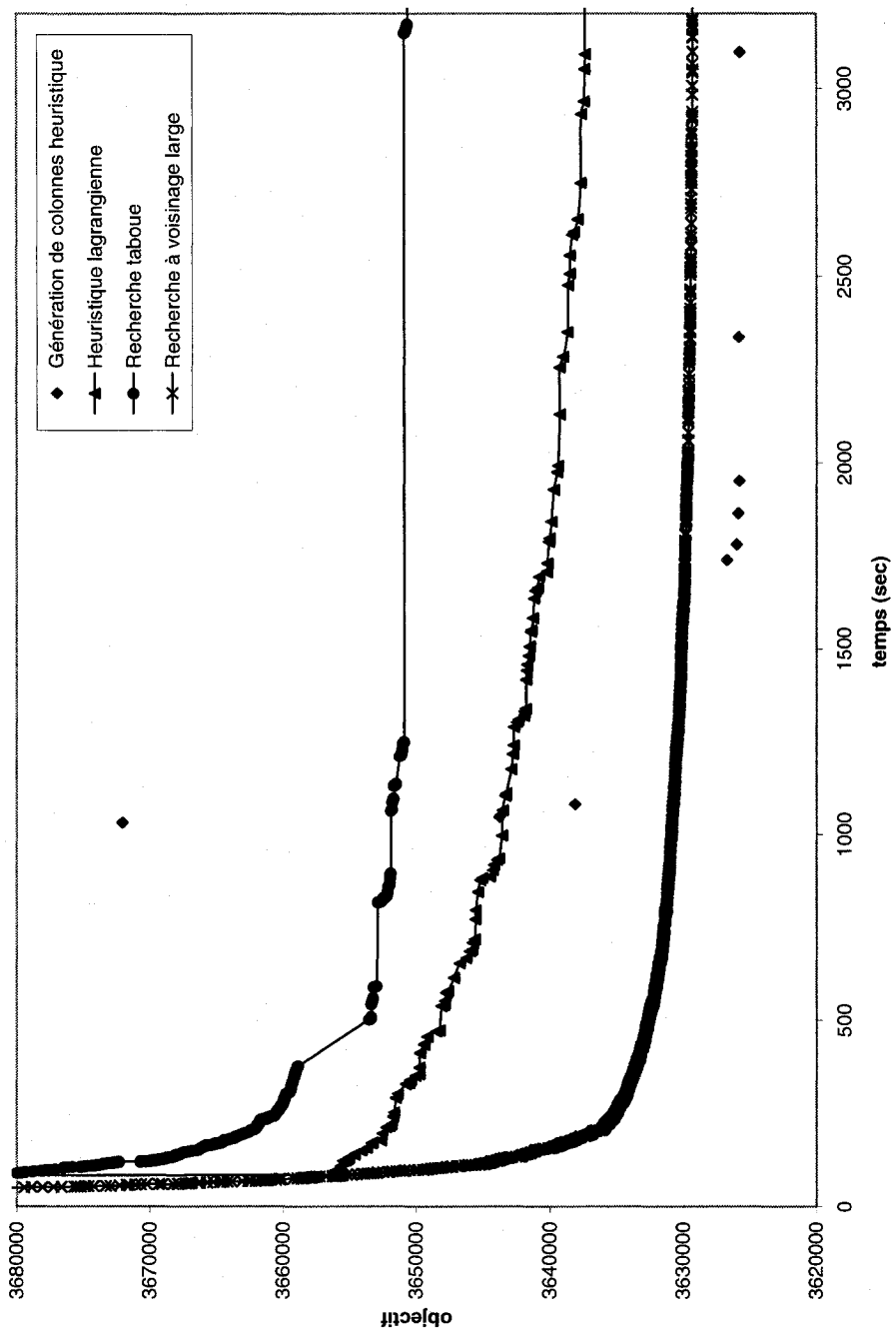

Figure 2.7 – Comparaison des heuristiques pour 1500 tâches 8 dépôts

Chapitre 3 : Améliorations

À partir des méthodes présentées dans le chapitre précédent, nous avons implémenté différentes améliorations. Nous avons tout d'abord tenté d'obtenir des solutions entières de trois façons pour chaque noeud de branchement de la méthode de génération de colonnes. Nous avons également, pour les méthodes de génération de colonnes et de SEP, diminué le nombre d'arcs conservés lors d'un pré-traitement afin de diminuer la taille des problèmes, et donc les temps de calcul. Tous les tests de ce chapitre ne concernent que les instances avec 4 dépôts. Pour la recherche de solutions entières, seules les instances de 1000 et 1500 tâches sont utilisées car les instances de 500 tâches n'ont pas assez de noeuds dans l'arbre de branchement pour permettre une bonne analyse.

3.1 Recherche de solutions entières

Cette stratégie est justifiée par le fait que la méthode de génération de colonnes ne fournit une solution entière qu'à la toute fin du processus. Tout au long de la résolution, différents noeuds de l'arbre de branchement sont visités mais ne fournissent que des solutions fractionnaires (sauf au dernier noeud). Nous avons donc cherché à obtenir, à chacun de ces noeuds, une solution entière.

À partir de la solution fractionnaire disponible à la fin d'un noeud de l'arbre de branchement, les colonnes dont le flot est supérieur à 0,7 sont fixées à 1. Certaines tâches se retrouvent donc déjà couvertes par ces tournées fixées. Trois approches ont été implémentées pour compléter la solution entière en assignant les tâches restantes :

résoudre un SDVSP, appliquer quelques itérations de recherche taboue à cette solution et résoudre un problème de partitionnement d'ensemble. Suite aux calculs effectués, le processus de branchement continue comme avant, sans tenir compte de la solution entière trouvée.

3.1.1 Résoudre un SDVSP

Une fois les tâches appartenant aux tournées dont le flot est supérieur ou égal à 0,7 fixées et enlevées, on peut obtenir une solution entière en procédant selon la méthode décrite dans la section 2.4.3, c'est-à-dire que l'on résout un SDVSP pour les tâches et les véhicules restants et que les tournées ainsi obtenues sont réassignées aux dépôts multiples de façon gloutonne. Jumelées avec les tournées déjà fixées, une solution entière est obtenue.

3.1.2 Appliquer une recherche taboue

Nous avons tenté d'améliorer les solutions obtenues par l'approche présentée à la sous-section 3.1.1 en appliquant, à chaque solution trouvée, quelques itérations de la recherche taboue présentée à la section 2.4. Après analyse, nous avons choisi d'effectuer 25 itérations de la méthode taboue. Il s'agit d'un bon compromis entre la qualité des solutions et l'augmentation des temps de calcul. Les courbes obtenues par ces deux approches sont présentées dans les figures 3.1 et 3.2 pour la moyenne des 5 instances de 1000 tâches et 1500 tâches, respectivement.

Ces courbes fournissent des points intermédiaires pour la résolution par la méthode de génération de colonnes avec $Q = 0$, représentée par le point noir le plus à droite du graphique. On peut donc remarquer que l'on arrive à obtenir des solutions entières

deux fois plus rapidement. On ne peut malheureusement pas faire mieux à moins d'accélérer la résolution de la relaxation linéaire. De plus, la qualité des solutions n'est pas très bonne si on la compare à celle obtenue par la RVL.

3.1.3 Résoudre un problème de partitionnement d'ensemble

À chaque noeud de l'arbre de branchement, le problème maître du MDVSP est résolu par génération de colonnes. Les colonnes (ou tournées) dont le flot est supérieur à 0,7 sont fixées à 1. Un algorithme de SEP (CPLEX 9.0.1) est appliqué afin de résoudre un problème de partitionnement d'ensemble. Ce problème est construit avec les colonnes contenant les tâches restantes afin de sélectionner un sous-ensemble de ces colonnes permettant de couvrir toutes ces tâches. Nous limitons la recherche au premier sous-ensemble réalisable trouvé. Nommons *Méthode 1* cette méthode pour références futures.

Les résultats ont permis de constater que le nombre de colonnes disponibles à chaque noeud de l'arbre de branchement n'est souvent pas assez grand. En effet, il arrive très souvent que l'algorithme de SEP ne trouve pas de solutions entières réalisables. Plus précisément, pour les instances de 1000 tâches, $32,91\%$ des noeuds, en moyenne, fournissent une solution entière. Pour les instances de 1500 tâches, ce nombre passe à $16,88\%$. De plus, le temps total requis avant la première solution de qualité est supérieur au temps pris par la méthode de génération de colonnes ($179s$ de plus pour 1000 tâches et $864s$ de plus pour 1500 tâches). Ceci est causé par le temps passé à chercher une solution entière (limité à 10 secondes pour chaque noeud). En allouant moins de temps à la méthode de SEP, c'est le pourcentage de réussite qui chute.

Pour palier à ce problème, nous avons choisi d'augmenter le nombre de colonnes conservées dans le problème maître (*Méthode 2*). Dans la méthode implémentée,

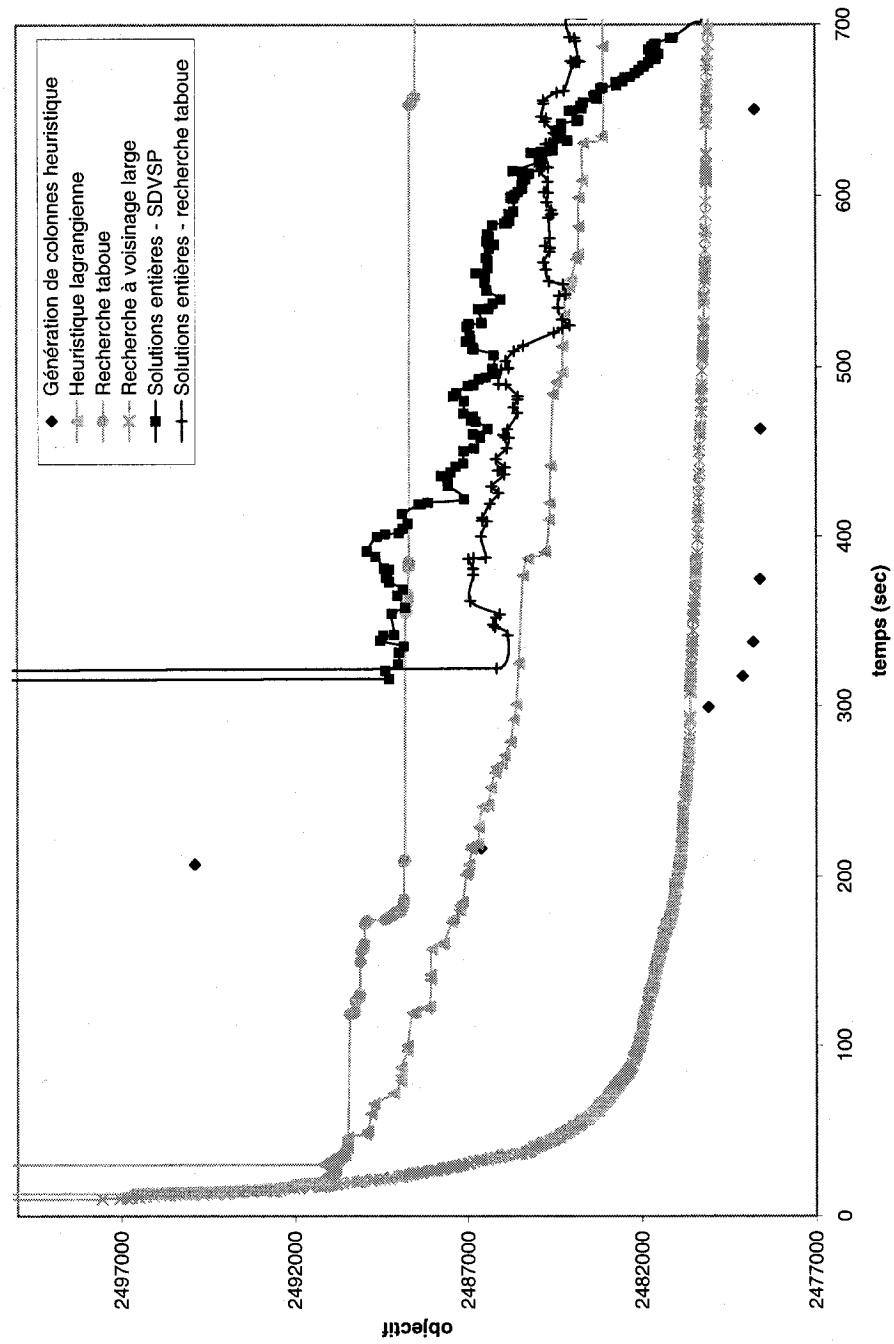

Figure 3.1 – Génération de solutions entières pour 1000 tâches

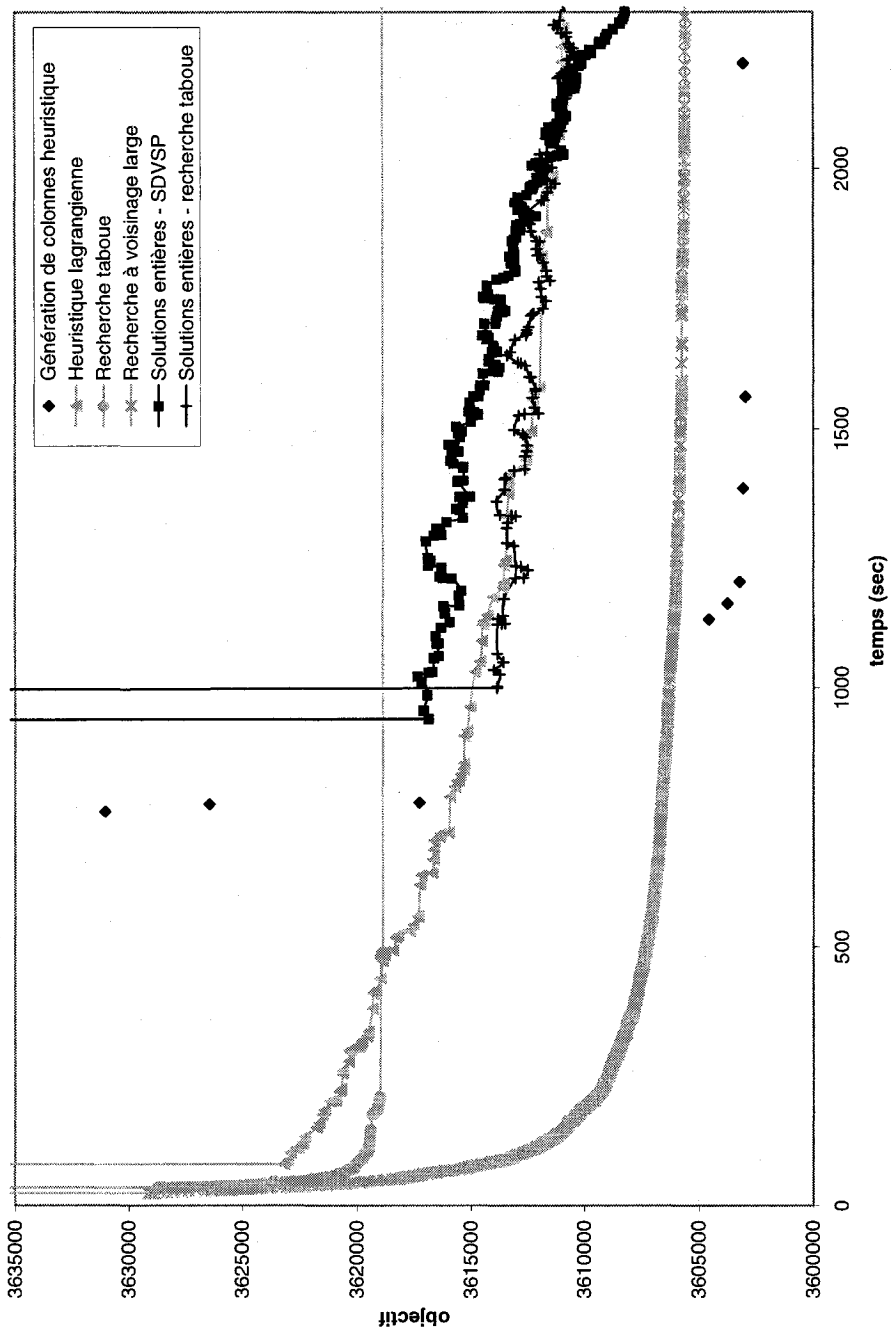

Figure 3.2 – Génération de solutions entières pour 1500 tâches

quand il y a trop de colonnes, on en enlève pour accélérer la résolution. Ceci se fait à l'aide de deux paramètres : un pour gérer le nombre de colonnes maximal permis et un autre pour indiquer le nombre de colonnes à retirer. En doublant le premier paramètre et en divisant par deux le second, nous avons réussi à obtenir des pourcentages de $40,92\%$ pour 1000 tâches et de $19,44\%$ pour 1500 tâches. Les temps de calcul sont toujours très longs, soit environ $163s$ (1000 tâches) et $789s$ (1500 tâches) de plus que le temps total pris par la génération de colonnes.

Finalement, nous avons tenté de conserver uniquement un sous-ensemble de colonnes dont le coût réduit est petit, c'est-à-dire inférieur à 100 dans notre cas (*Méthode 3*). Nous espérions ainsi accélérer la résolution par la méthode de SEP. Les résultats ont été décevants une fois de plus : les temps de calcul se sont améliorés de très peu. Par rapport à la *Méthode 2*, $63s$ ont été sauvées en moyenne pour les instances de 1000 tâches et $159s$ pour celles de 1500 tâches.

Le tableau 3.1 résume les résultats des tests (moyennes sur les 5 instances) pour la génération de solutions entières avec la résolution d'un problème de partitionnement d'ensemble. La colonne *Solutions trouvées* donne le pourcentage de noeuds où une solution entière a été trouvée. La colonne *Temps supplémentaire* indique le pourcentage de temps supplémentaire pris par les 3 méthodes pour avoir une première solution entière par rapport au temps total requis par la méthode de génération de colonnes pour fournir sa solution.

Le tableau 3.2 montre ce qui se passe pour chaque instance de 1000 tâches dans la *Méthode 1*. Il donne le nombre de noeuds de l'arbre de branchement, le nombre de noeuds où une solution entière est trouvée et le numéro du noeud où la première solution entière est trouvée. Pour cette première solution trouvée, on a la différence de temps, en pourcentage, pris pour la trouver (par rapport au temps total de la méthode de génération de colonnes), la différence de coût, en pourcentage, toujours

Tableau 3.1 – Résultats pour la génération de solutions entières par résolution d'un problème de partitionnement d'ensemble

		Solutions trouvées (%)	Temps supplémentaire (%)
1000 tâches	Méthode 1	32,91	27,57
	Méthode 2	40,92	25,05
	Méthode 3	48,23	15,27
1500 tâches	Méthode 1	16,88	39,24
	Méthode 2	19,44	35,82
	Méthode 3	22,75	28,58

par rapport au coût de la méthode de génération de colonnes sans recherche de solutions entières et le nombre de tâches restantes lorsque cette première solution est trouvée. Les résultats des autres méthodes sont similaires. On peut remarquer que les premiers noeuds ne fournissent pas souvent de solutions entières. Beaucoup de temps est donc perdu pour ces premiers noeuds. Une variation de cette approche pourrait être d'attendre quelques noeuds avant d'appliquer la procédure de SEP ou d'attendre qu'il reste moins de tâches à fixer. Moins il reste de tâches à fixer, plus les chances d'obtenir un sous-ensemble de colonnes fournissant une solution entière sont grandes. Par contre, le but d'obtenir des solutions entières plus rapidement n'est pas vraiment atteint.

Tableau 3.2 – Résultats de la *Méthode 1* sur chaque instance de 1000 tâches

Instance	Noeuds	Solutions trouvées	Première solution	Temps (%)	Coût (%)	Tâches restantes
0	56	20	30	40,94	-0,21	523
1	27	10	11	19,41	0,29	659
2	41	11	15	8,63	-0,13	686
3	27	5	10	17,67	0.01	743
4	28	13	4	20,42	0.17	790

3.2 Réduction du nombre d'arcs

La formulation avec connexions explicites basée sur les arcs relie par un arc toutes les tâches (représentées par des noeuds) pouvant être effectuées successivement par un même véhicule. Pour chaque noeud, il peut donc y avoir $n-1$ arcs inter-tâches sortants. On peut supposer que, parmi tous les arcs sortants, ceux de coût très élevé ont peu de chance de faire partie de la solution optimale. Nous avons donc modifié le générateur d'instances de façon à ce que les arcs de coût élevé soient éliminés. Nous avons effectué des tests en conservant au maximum les n/α arcs sortant de chaque noeud, α entre 2 et 7. Nos tests ont permis de confirmer notre hypothèse que, jusqu'à un certain point, la résolution des problèmes comportant moins d'arcs est plus rapide tout en conservant la qualité des solutions initialement obtenues. Ceci est valide pour les deux méthodes testées soit, les méthodes de génération de colonnes et de SEP.

Le tableau 3.3 présente les résultats pour les trois tailles de problèmes et pour les différentes valeurs de α. Les instances ont été résolues par génération de colonnes, sans arrêter prématurément la résolution du problème maître ($Q = 0$). Les valeurs du tableau correspondent à la différence, en pourcentage, avec les valeurs obtenues par génération de colonnes au chapitre 2 lorsque $Q = 0$ et $\alpha = 1$ (conservation de tous les arcs, sans arrêt prématuré). En fixant à $0,05\%$ la détérioration acceptable de la valeur de la solution, on peut remarquer que, pour les instances de 500 tâches, $\alpha = 3$ détermine le seuil au-delà duquel la rapidité commence à nuire à la qualité de la solution. De la même façon, on remarque que $\alpha = 5$ pour 1000 tâches et $\alpha = 6$ pour 1500 tâches déterminent les seuils limites.

Cette stratégie d'accélération, consistant à diminuer le nombre d'arcs, accélère la résolution des sous-problèmes. Le tableau 3.4 indique la proportion du temps (moyenne) passé dans le problème maître (PM) et dans les sous-problèmes (SP) pour les instances de chaque taille, avec ou sans réduction du nombre d'arcs. On peut remarquer

Tableau 3.3 – Différences en pourcentage avec moins d'arcs (génération de colonnes)

	500 tâches		1000 tâches		1500 tâches	
Nombre d'arcs	temps	valeur	temps	valeur	temps	valeur
$n/2$	-18,0340	-0,0009	-11,7398	-0,0035	-3,1712	0,0002
$n/3$	**-24,1046**	**0,0008**	-19,0015	0,0007	-16,3934	-0,0018
$n/4$	-41,7831	0,1484	-25,1437	-0,0036	-23,0404	0,0479
$n/5$	-50,8001	0,1493	**-28,1997**	**0,0039**	-30,2696	0,0478
$n/6$	-58,9281	0,2948	-36,7625	0,0754	**-34,8025**	**0,0493**
$n/7$	-60,6807	0,4545	-38,6384	0,0842	-36,3612	0,0576

Tableau 3.4 – Proportion du temps passé dans le problème maître et les sous-problèmes

	500 tâches		1000 tâches		1500 tâches	
	$\alpha = 1$	$\alpha = 3$	$\alpha = 1$	$\alpha = 5$	$\alpha = 1$	$\alpha = 6$
SP	84,3439%	77,5404%	58,1750%	37,4719%	56,3823%	32,2338%
PM	15,1771%	21,9514%	41,5277%	62,2567%	43,3905%	67,5697%

que le temps passé au niveau des sous-problèmes a diminué dans les trois cas lorsque le nombre d'arcs est réduit. De plus, avant la réduction ($\alpha = 1$), on peut remarquer que plus la taille des instances est grande, plus le temps passé à résoudre les sous-problèmes est petit par rapport au problème maître. Il est donc normal d'obtenir, dans le tableau 3.3, des gains en temps de calcul plus grands pour les instances de 500 tâches que pour celles de 1000 ou 1500 tâches (pour une même valeur de α).

Le tableau 3.5 présente les pourcentages résultant de la résolution par SEP. Ils sont similaires à ceux obtenus par génération de colonnes. On remarque cependant une instabilité au niveau de l'heuristique de CPLEX pour les instances de 1500 tâches. En effet, pour 4 des 6 valeurs de α testées, les temps de calcul sont plus longs que lorsque tous les arcs sont conservés, comme si le logiciel peinait à trouver une solution entière.

Tableau 3.5 – Différences en pourcentage avec moins d'arcs (SEP)

	500 tâches		1000 tâches		1500 tâches	
Arcs	temps	valeur	temps	valeur	temps	valeur
$n/2$	-13,2956	0,0001	-0,2052	0,0005	93,0812	0,0573
$n/3$	**-31,2294**	**-0,0039**	-13,2497	0,0045	51,3460	0,0595
$n/4$	-20,5699	0,1545	-49,4980	0,0038	3,1800	0,0795
$n/5$	-46,4144	0,1541	**-59,6574**	**0,0034**	97,9771	0,0542
$n/6$	-54,4451	0,2969	-65,7775	0,0870	**-49,8259**	**0,0489**
$n/7$	-61,0232	0,4433	-70,6761	0,0820	-37,4962	0,1097

À partir de la meilleure valeur de α trouvée pour chaque taille de problème, une courbe de points peut être tracée où chaque point correspond à une résolution du problème par génération de colonnes en arrêtant prématurément la résolution du problème maître (comme dans la section 2.6). Les figures 3.3 à 3.5 présentent ces nouvelles courbes. Le point correspondant à la meilleure valeur de α pour la méthode de SEP apparaît également sur la figure, ce qui n'était pas le cas précédemment (pour 1000 et 1500 tâches) dû au temps de calcul trop long. La courbe de la recherche à voisinage large, de même que la courbe de la génération de colonnes heuristique originale sont présentes en pâle dans le graphique pour mieux visualiser les améliorations apportées par la nouvelle courbe en plus foncé.

Nous aurions également pu appliquer cette idée sur les autres méthodes présentées au chapitre 2, comme la recherche à voisinage large. Cependant, il aurait été trop coûteux en temps de développement de faire ces tests à cause de la façon dont ces méthodes ont été implémentées.

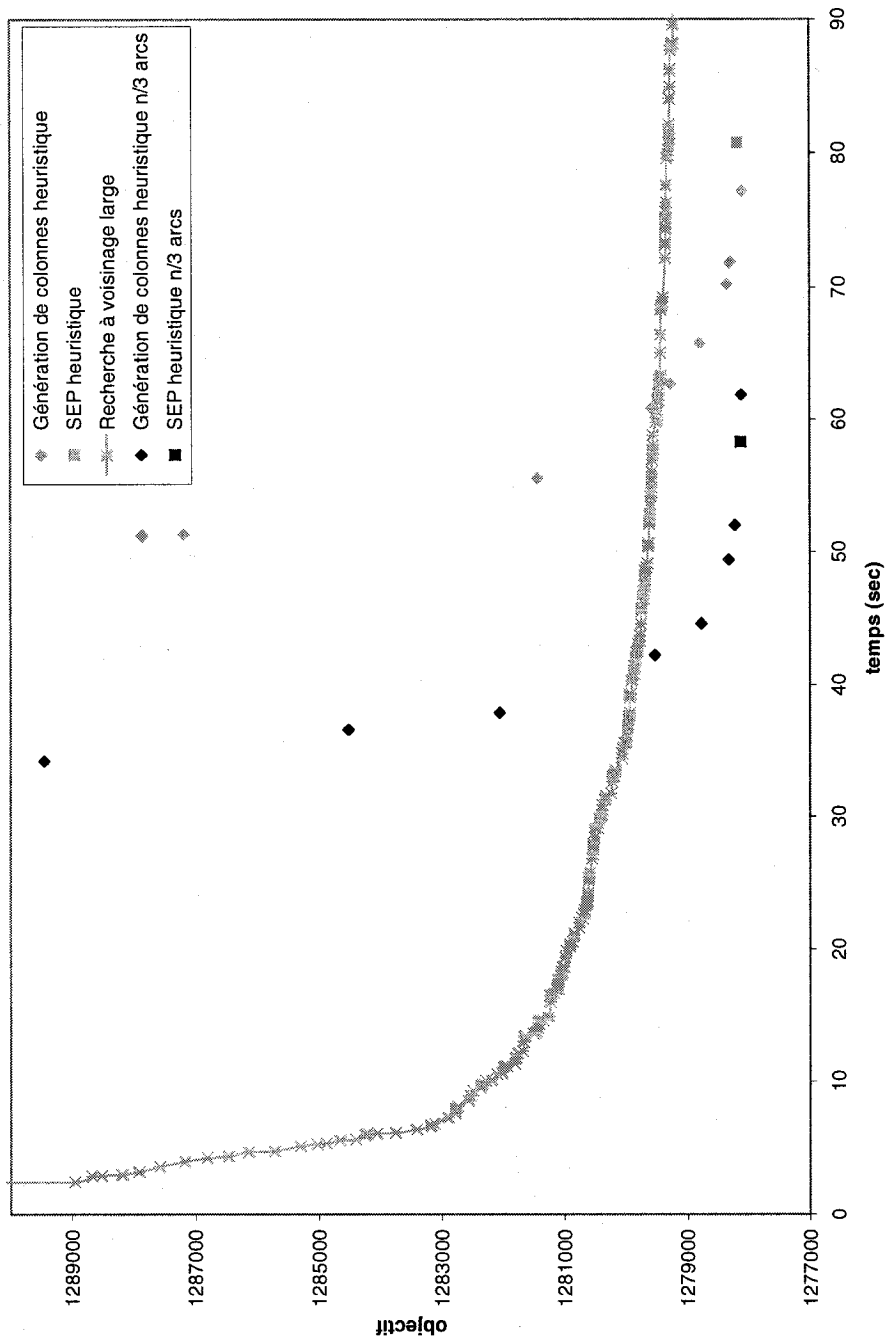

Figure 3.3 – Réduction du nombre d'arcs pour 500 tâches ($\alpha = 3$)

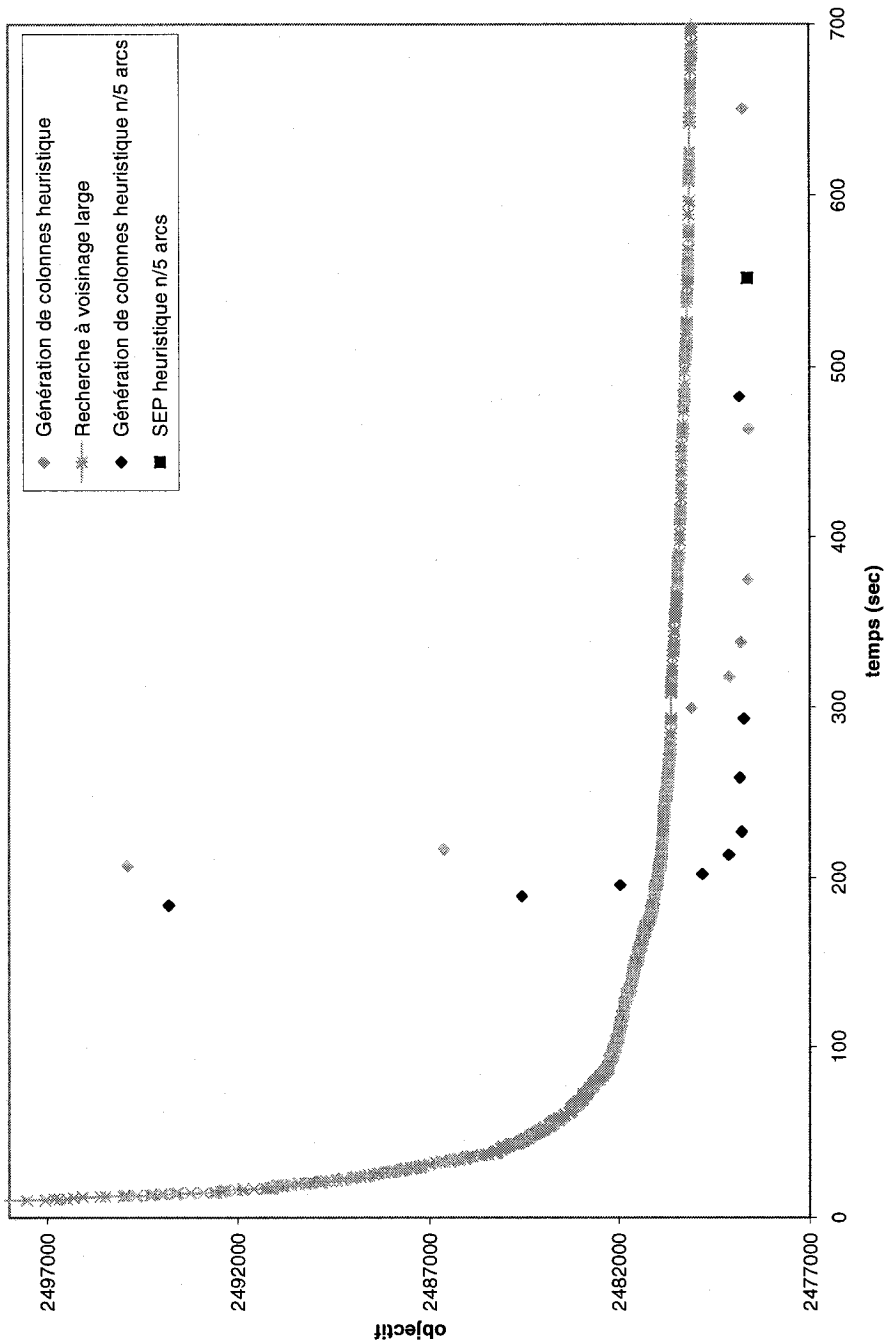

Figure 3.4 – Réduction du nombre d'arcs pour 1000 tâches ($\alpha = 5$)

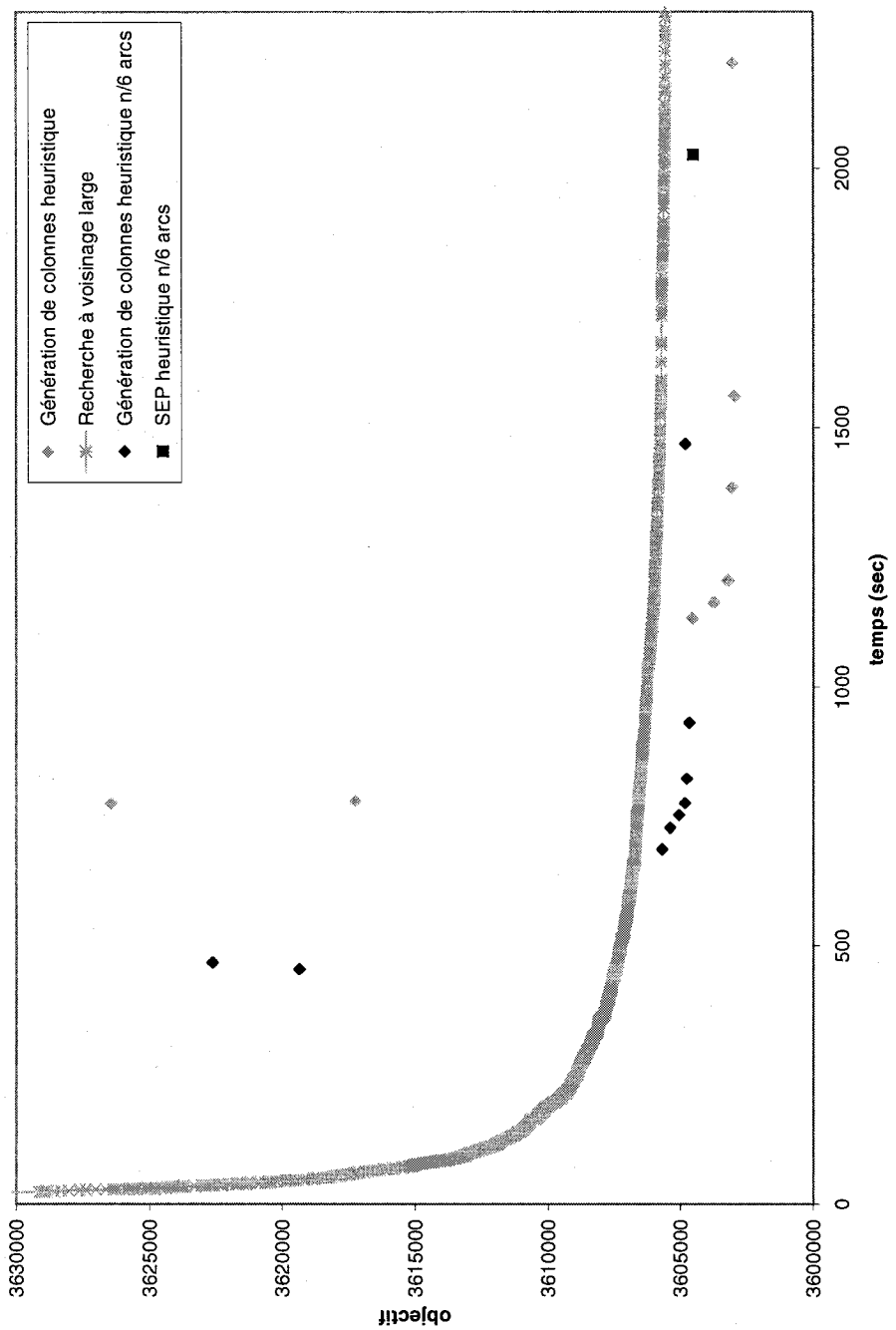

Figure 3.5 – Réduction du nombre d'arcs pour 1500 tâches ($\alpha = 6$)

Conclusion

Ce projet de maîtrise avait pour but de comparer différentes approches heuristiques basées sur les techniques de programmation mathématique avec des méthodes métaheuristiques appliquées au problème d'horaires de véhicules avec plusieurs dépôts. L'approche de génération de colonnes imbriquée dans une procédure de séparation et évaluation progressive nous a fourni les solutions ayant le côut le plus faible pour presque toutes les tailles de problème testées. La méthode de SEP donne des solutions de meilleur coût dans certains cas, mais celles-ci sont trouvées en des temps deux fois plus long s'il y a 4 dépôts ou six fois plus longs s'il y a 8 dépôts.

On peut cependant obtenir de bonnes solutions dans des temps beaucoup plus courts en utilisant une recherche à voisinage large ou, pour les instances de 500 tâches, une heuristique lagrangienne. La recherche taboue, quant à elle, semble éprouver certaines difficultés à se sortir des minima locaux et ne permet pas d'être concurrentielle avec les autres méthodes. L'approche de séparation et évaluation progressive heuristique subit les mêmes conclusions, dû aux temps de calcul trop longs.

Suite à ces premiers résultats, nous avons tenté d'améliorer les temps de calcul pour les méthodes de génération de colonnes heuristique et de séparation et évaluation progressive heuristique en éliminant d'entrée de jeu les arcs ayant un coût très élevé. Nous avons ainsi pu obtenir des solutions dont la qualité ne s'est pas détériorée de plus de 0,05% de sa valeur originale avec des gains de temps de l'ordre de 24,1% à 34,8% pour la résolution par génération de colonnes et de 31,2% à 59,6% pour l'algorithme de séparation et évaluation progressive.

L'autre amélioration proposée, consistant à générer des solutions entières à chaque noeud de l'arbre de branchement de la méthode de génération de colonnes, n'a pas

permis l'obtention de bonnes solutions plus rapidement car le temps passé à résoudre la relaxation linéaire du problème (donc avant d'obtenir la première solution entière) est trop long et la solution obtenue n'est pas de bonne qualité.

Cette étude pourrait se poursuivre de différentes manières. Tout d'abord, la méthode de recherche taboue peut être améliorée en définissant un voisinage plus approprié au problème qui permettrait de ne pas rester bloqué aux minimums locaux. De plus, les résultats obtenus ne sont pas nécessairement valides pour d'autres types ou d'autres tailles de problèmes. Il pourrait être intéressant d'effectuer une comparaison semblable sur de très grosses instances ou sur d'autres problèmes, comme celui de tournées de véhicules avec contraintes de capacité et fenêtres de temps. On pourrait également approfondir la recherche de solutions entières intermédiaires pour la méthode de génération de colonnes. Nous n'avons, en effet, qu'explorer quelques-unes des possibilités.

Finalement, nous avons conclu que la réduction du nombre d'arcs était très avantageuse car elle permet de diminuer les temps de calcul sans détériorer la qualité des solutions. Étant donné la façon dont ont été implémentées les autres méthodes, nous n'avons pas testé cette idée sur ces dernières. Une étude pourrait être faite afin de vérifier si les résultats sont similaires.

Bibliographie

AHUJA, R.K., T.L. MAGNANTI ET J.B. ORLIN (1993). *Networks Flows : Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey.

AHUJA, R.K., O. ERGUN, J.B. ORLIN ET A.P. PUNNEN (2002). *A Survey of Very Large-Scale Neighborhood Search Techniques, Discrete Applied Mathematics 123*, 75–102.

BIANCO, L., A. MINGOZZI ET S. RICCIARDELLI (1995). An Exact Algorithm for Combining Vehicle Trips, dans : J.R. Daduna, I. Branco and J. Paixão (éds), *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems 430, Springer-Verlag, Berlin, 145–172.

BIANCO, L., A. MINGOZZI ET S. RICCIARDELLI (1994). A Set Partitioning Approach to the Multiple Depot Vehicle Scheduling Problem, *Optimization Methods and Software 3*, 163–194.

BRANCO, I., A. COSTA ET PAIXÃO, J.M.P. (1995). Vehicle Scheduling Problem with Multiple Type of Vehicles and a Single Depot, dans : Daduna *et al.* (éds), *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, 115–129.

BODIN, L., B. GOLDEN, A. ASSAD ET M. BALL (1983). Routing and Scheduling of Vehicles and Crews : The State of the Art, *Computers and Operations Research 10*, 63–211.

BODIN, L., D. ROSENFIELD ET A. KYDES (1978). UCOST : A Micro Approach to a Transit Planning Problem, *Journal of Urban Analysis 5*, 47–69.

CARPANETO, G., M. DELL'AMICO, M. FISCHETTI ET P. TOTH (1989). A Branch and Bound Algorithm for the Multiple Vehicle Scheduling Problem. *Networks 19*, 531–548.

CORDEAU, J.-F., M. GENDREAU, A. HERTZ, G. LAPORTE ET J.-S. SORMANY (2004). New Heurisitcs for the Vehicle Routing Problem. *Les cahiers du GERAD G-2004-33*, École des Hautes Études Commerciales, Montréal, Canada.

CORDEAU, J.-F., G. LAPORTE ET A. MERCIER (2001). A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. *Journal of the Operational Research Society 52*, 928–936.

CORDEAU, J.-F., M. GENDREAU ET G. LAPORTE (1997). A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems, *Networks 30*, 105–119.

DANTZIG, G.B. ET P. WOLFE (1960). Decomposition Principle for Linear Programs. *Operations Research 8*, 101–111.

DELL'AMICO, M., M. FISCHETTI ET P. TOTH (1993). Heuristic Algorithms for the Multiple Depot Vehicle Scheduling Problem, *Management Science 39*, 115–125.

DEREU, G. (2004). Comparaison d'une Méthode de Génération de Colonnes et d'une Méthode de Recherche Taboue pour le Problème d'Horaires de Véhicules avec Dépôts Multiples, *Mémoire de maîtrise*, École Polytechnique de Montréal, Canada.

DESAULNIERS, G. ET M. HICKMAN (2003). Public Transit, Les cahiers du Gerad G–2003–77, École des Hautes Études Commerciales, Canada, *à paraître dans : Handbooks in OR&MS, Volume on Transportation*, G. Laporte et C. Barnhart (éds).

DESAULNIERS, G., J. LAVIGNE ET F. SOUMIS (1998). Multi-Depot Vehicle Scheduling Problems with Time Windows and Waiting Cost. *European Journal of Operational Research 111*, 479–494.

DESROSIERS, J., Y. DUMAS, M.M. SOLOMON ET F. SOUMIS (1995). Time Constrained Routing and Scheduling, dans : M.O. Ball *et al.* (éds), *Network Routing*, Handbooks in Operations Research and Management Science 8, Elsevier Science, Amsterdam, 35–139.

FORBES, M.A., J.N. HOLT ET A.M. WATTS (1994). An Exact Algorithm for Multiple Depot Bus Scheduling, *European Journal of Operational Research 72*, 115–124.

FRELING, R., J.M.P., PAIXÃO ET A.P.M. WAGELMANS (2001). Models and Algorithms for Single-Depot Vehicle Scheduling, *Transportation Science 35*, 165–180.

GILMORE, P.C. ET R.W. GOMORY (1961). A Linear Programming Approch to the Cutting Stock Problem. *Operations Research 9*, 849–859.

GINTNER V., N. KLIEWER ET L. SUHL (2005). Solving Large Multiple-Depot Multiple-Vehicle-Type Bus Scheduling Problems in Practice, *OR Spectrum 27*, 507–523.

HAASE, K., G. DESAULNIERS ET J. DESROSIERS (2001). Simultaneous Vehicle and Crew Scheduling in Urban Mass Transit Systems, *Transportation Science 35(3)*, 286–303.

HADJAR, A., O. MARCOTTE ET F. SOUMIS (2006). A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem, *Operations Research 54(1)*, 130–149.

KLIEWER N., T. MELLOULI ET L. SUHL (2005). A Time-Space Network Based Exact Optimization Model for Multi-Depot Bus Scheduling, *à paraître dans : European Journal of Operational Research.*

KOKOTT A. ET A. LÖBEL (1996). Lagrangean Relaxations and Subgradient Methods for Multiple-Depot Vehicle Scheduling Problems, Technical Report SC 96-22, Konrad-Zuse-Zentrum fur Informationstechnik, Berlin.

LAMATSCH, A. (1992). An Approach to Vehicle Scheduling with Depot Capacity Constraints, dans : M. Desrochers and J.-M. Rousseau (éds), *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems 386, Springer-Verlag, Berlin, 181–195.

LÖBEL, A. (1998). Vehicle Scheduling in Public Transit and Lagrangean Pricing, *Management Science 44(12)*, 1637–1649.

LÖBEL, A. (1997). Optimal Vehicle Scheduling in Public Transit, Ph.D. thesis, Technische Universität Berlin, Germany.

MARIN, J.-M. (2005). Stratégies d'accélération pour le problème de tournées de véhicules avec dépôts multiples, *Mémoire de maîtrise*, École Polytechnique de Montréal, Canada.

MESQUITA, M. ET J. PAIXÃO (1999). Exact Algorithms for the Multi-Depot Vehicle Scheduling Problem Based on Multicommodity Network Flow Type Formulations, dans : N.H.M. Wilson (éds), *Computer-Aided Transit Scheduling*, Springer Verlag, Berlin, 223–246.

MESQUITA, M. ET J. PAIXÃO (1992). Multiple Depot Vehicle Scheduling Problem : A New Heuristic Based on Quasi-Assignment Algorithms, dans : M. Desrochers and J.M. Rousseau (éds), *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems 386, Springer Verlag, Berlin, 167–180.

ODONI, A.R., J.-M. ROUSSEAU ET N.H.M. WILSON (1994). Models in Urban and Air Transportation, dans : S.M. Pollock *et al.* (éds), *Operations Research and the Public Sector*, Handbook in Operations Research and Management Science 6, North-Holland, Amsterdam, 107–150.

OUKIL, A., H. BEN AMOR, J. DESROSIERS ET H. EL GUEDDARI (2007). Stabilized Column Generation for Highly Degenerate Multiple-Depot Vehicle Scheduling Problems, *Computers & Operations Research 34*, 817–834. *À paraître.*

RIBEIRO, C. ET F. SOUMIS *(1994). A Column Generation Approach to the Multiple Depot Vehicle Scheduling Problem.* Operations Research 42(1), *41–52.*

ROPKE, S. ET D. PISINGER *(2004). An Adaptative Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows, rapport de recherche, Université de Copenhague.*